ПРИЛОЖЕНИЕ А

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

ОБРАБОТКА МАССИВОВ СТРУКТУРИРОВАННЫХ ДАННЫХ «ВОКЗАЛ»

ТЕКСТ ПРОГРАММЫ

КР.АС59.200054

Руководитель И.Н. Аверина

Выполнил
студент 1 курса
группы АС-59 Д.С.Качан

2021

СОДЕРЖАНИЕ

proga.cpp – основной модуль, в котором содержится функция, которая вызывается в начале программы, меню для работы с программой и чтение данных из файла.

train.h – файл, содержащий структуру «Поезд» и структуру буфера.

train_func.h, train_func.cpp – пользовательская библиотека, в которой содержатся функции для работы с массивом записей: функция вывода записей на экран, функция добавления записи, функция редактирования записи, функция удаления записей, функция сортировки записей, функция выборки записей, функция отмены действия.

menu.h, menu.cpp – пользовательская библиотека, в который содержатся основные функции, вывода меню для различных типов работы функций: функция вывода главного меню, функция вывода меню сортировки, функция вывода меню обратной сортировки, функция вывода меню сохранения в файл, функция вывода меню удаления записей, функция вывода меню выборки записей, функция вывода меню изменения записи.

file.h, file.cpp – пользовательская библиотека, в которой содержатся все функции, относящиеся к работе с файлами: чтение и запись в файл, создание индексных файлов.

clear.h, clear.cpp, clear_w.cpp – пользовательская библиотека, в которой сожержится функция для очистки экрана.

config_func.h, config_func.cpp – пользовательская библиотека, содержащая функции для работы с памятью: добавление и удаление ячейки памяти, сохранение массива в буфер для отмены.

delete_func.h, delete_func.cpp – пользовательская библиотека, в которой сожержатся функции для удаления записей.

sort_func.h, sort_func.cpp – пользовательская библиотека, в которой сожержатся функции для сортировки записей.

select_func.h, select_func.cpp – пользовательская библиотека, в которой сожержатся функции для выборки записей.

proga.cpp

```
/*
Главный файл программы
Сюда подключены остальные файлы
*/

#include "train.h"
#include "train_func.h"
#include "file.h"
#include "menu.h"
#include "clear.h"

#include <iostream>
#include <fstream>
#include <span>


int main()
{
    int train_count = 0; // Количество поездов на данный момент
    Train* train_station = new Train[train_count]; // Массив структур

    int train_buffer_count = 0; // Количество действий в буфере
    TrainBuffer* trains_buffer = new TrainBuffer[10]; // Буфер для хранения последних
        действий

    // Массив указателей на функции
    void(*menu_func_arr[])(Train*&, TrainBuffer*, int&, int&) = {print_train, add_train,
        change_train, delete_train, sort_train, select_train, undo_action};

    int is_file_not_open = 1; // Флаг состояния открытия файла (изначально не открыт)
    char file_name[256]; // Строка с именем файла
    clear();
    while (is_file_not_open) { // Цикл открытия файла
        std::cout << "Введите имя файла: ";
        std::cin.getline(file_name, 256); // Ввод имени файла
        is_file_not_open = get_records_from_file(file_name, train_station, train_count); //
            Получение записей из файла и флага состояния
        if (is_file_not_open) { // Если файл не открыт
            bool is_false = true;
            while (is_false) {
                int create;
                std::cout << "Создать новый файл? (0 - не         создавать, 1 - создать): ";
                std::cin >> create;
                if (create < 0 || create > 1) {
                    std::cout << "Неправильный ввод!!!";
                    is_false = true;
                }
                else {
                    if (create == 0) {
                        is_false = false;
                        std::cin.ignore(32767, '\n');
                    }
```

```
            else if (create == 1) {
                is_false = false;
                std::ofstream file(file_name, std::ios::binary);
                file.close();
                is_file_not_open = get_records_from_file(file_name, train_station,
                    train_count);
            }
        }
    }
}

create_index_file(train_station, train_count); // Создание индексных файлов

bool is_working = true; // Флаг работы программы
while (is_working) { // Цикл работы программы
    print_menu(); // Вывод меню действий на экран
    int action; // Переменная действия
    while (true) {
        std::cout << "Выберите действие: ";
        std::cin >> action;

        if (std::cin.fail()) { // Если ввод был неудачным
            std::cin.clear();
            std::cin.ignore(32767, '\n');
            std::cout << "Неправильный ввод!!!" << std::endl;
        }
        else {
            std::cin.ignore(32767, '\n');
            break;
        }
    }

    clear();

    if (action >= 0 and action <= 7) { // Проверка действия
        if (action == 0) { // Отмена
            is_working = false;
        }
        else {
            menu_func_arr[action - 1](train_station, trains_buffer, train_count,
                train_buffer_count); // Вызов функции
        }
    }
    else {
        std::cout << "Такого действия нет" << std::endl;
    }

    put_records_in_file(file_name, train_station, train_count); // Запись данных в файл
}

delete[] train_station; // Очистка пямяти
```

```
    for (auto buffer: std::span(trains_buffer, train_buffer_count)) { // Очистка буфера
        отмены
        delete[] buffer.trains;
    }
    delete[] trains_buffer;
    return 0;
}
```

train.h

```
#ifndef TRAIN_H
#define TRAIN_H


// Структура "Поезд"
#pragma pack(push, 1)
struct Train {
    char number[5];
    char end_station[256];
    char days[64];
    char time_departure[7];
    char time_way[7];
    int stop_count;
};

// Структура буфера для отмены действий
struct TrainBuffer {
    int count;
    Train* trains;
};
#pragma pack (pop)

#endif
```

train_func.h

```
#ifndef TRAIN_FUNC_H
#define TRAIN_FUNC_H

#include "train.h"

void print_train(Train*&, TrainBuffer*,int&, int&);
void add_train(Train*&, TrainBuffer*, int&, int&);
void change_train(Train*&, TrainBuffer*, int&, int&);
void delete_train(Train*&, TrainBuffer*, int&, int&);
void sort_train(Train*&, TrainBuffer*, int&, int&);
void select_train(Train*&, TrainBuffer*, int&, int&);
void undo_action(Train*&, TrainBuffer*, int&, int&);

#endif
```

train_func.cpp

```
/*
Файл с функциями для работы с записями
*/

#include "train_func.h"
#include "delete_func.h"
#include "select_func.h"
#include "config_func.h"
#include "menu.h"
#include "file.h"
#include "clear.h"

#include <iostream>
#include <cstring>
#include <fstream>
#include <iomanip>
#include <span>


// Вывод всех поездов
// ========================================================================================
void print_train(Train*& trains, TrainBuffer* trains_buffer, int& count, int& buffer_count)
{
    std::cout << " Номер" << '|' << " Конечная станция" << '|' << " Дни следования" << '|'
        << " Отправление" << '|' << "Время в пути" << '|' << " Кол-во остановок" << '|' <<
        "Время в пути в сутках";
    std::cout << std::endl;

    for (auto train: std::span(trains, count)) { // Цикл по всем записям
        std::cout << std::setw(7) << train.number << '|';
        std::cout << std::setw(20) << train.end_station << '|';
        std::cout << std::setw(20) << train.days << '|';
        std::cout << std::setw(12) << train.time_departure << '|';
        std::cout << std::setw(12) << train.time_way << '|';
        std::cout << std::setw(17) << train.stop_count << '|';

        // Расчет поля время в пути в сутках
        char hours_str1[3] = " ", minutes_str1[3] = " "; // Промежуточные переменные для
            часов и минут
        int end_ch;
        double time_in_hours;

        for (int ch = 0; ch < strlen(train.time_way); ch++) { // Запись часов записи
            if (train.time_way[ch] != ':') { // Сравнение с двоеточием
                hours_str1[ch] = train.time_way[ch];
            }
            else { // Если равны, то записываем индекс
                end_ch = ch + 1;
                break;
            }
        }
```

```
        minutes_str1[0] = train.time_way[end_ch]; // Запись минут записи
        minutes_str1[1] = train.time_way[end_ch + 1];
        time_in_hours = float(std::stoi(hours_str1)) + float(std::stoi(minutes_str1)) / 60.0;
            // Подсчёт времени записи

        std::cout << std::setw(21) << time_in_hours / 24.;
        std::cout << std::endl;
    }
}


// Добавление нового поезда
// ========================================================================================
void add_train(Train* &trains, TrainBuffer* trains_buffer, int& count, int& buffer_count)
{
    print_train(trains, trains_buffer, count, buffer_count);
    bool is_false = true;
    save_trains_in_buffer(trains, trains_buffer, count, buffer_count);

    add_memory_train(trains, count);

    int pos; // Позиция для добавления

    while (true) {
        std::cout << "Введите позицию для добавления записи после указанной записи (-1 - в
            конец): ";
        std::cin >> pos;

        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(32767, '\n');
            std::cout << "Неправильный ввод!!!" << std::endl;
        }
        else {
            std::cin.ignore(32767, '\n');
            if (pos < -1) {
                std::cout << "Позиция для добавления должна быть больше или равна -1!!!" <<
                    std::endl;
            }
            else if (pos == -1 || pos > count) { // Проверка на добавление в конец
                pos = count;
                break;
            }
            else {
                for (int i = count - 1; i >= pos; i--) { // Сдвиг элементов на один вперед
                    trains[i + 1] = trains[i];
                }
                break;
            }
        }
    }

    while (is_false) { // Проверка на правильность ввода
```

```
    std::cout << "Номер поезда: ";
    char input_buffer[1024];
    std::cin.getline(input_buffer, 1024);

    is_false = false;
    if (strlen(input_buffer) > 4) {
        std::cout << "Номер поезда должен состоять из менее, чем 5 символов!!!" <<
            std::endl;
        is_false = true;
        continue;
    }
    else {
        for (int i = 0; i < strlen(input_buffer); i++) {
            if (int(input_buffer[i]) < 48 || int(input_buffer[i]) > 57) {
                is_false = true;
                std::cout << "Номер должен состоять из цифр!!!" << std::endl;
                break;
            }
        }
        if (is_false) {
            continue;
        }
        strcpy(trains[pos].number, input_buffer);
    }
    for (int i = 0; auto train: std::span(trains, count)) { // Цикл по всем записям
        if (i == pos) continue;
        if (strcmp(trains[pos].number, train.number) == 0) { // Проверка на равенство
            номеров
            is_false = true;
            std::cout << "Такой номер уже есть в базе данных!!!" << std::endl;
        }
        i++;
    }
}

std::cout << "Конечная станция: ";
std::cin.getline(trains[pos].end_station, 256);

std::cout << "Дни следования: ";
std::cin.getline(trains[pos].days, 64);

is_false = true;
while (is_false) {
    std::cout << "Время отправления: ";
    std::cin.getline(trains[pos].time_departure, 7);

    char hours_str[3] = " ", minutes_str[3] = " "; // Промежуточные переменные для часов
        и минут
    int end_ch;

    for (int ch = 0; ch < strlen(trains[pos].time_departure); ch++) { // Запись часов
        записи
```

```
            if (trains[pos].time_departure[ch] != ':' && trains[pos].time_departure[ch] !=
                '.') { // Сравнение с двоеточием
                hours_str[ch] = trains[pos].time_departure[ch];
            }
            else { // Если равны, то записываем индекс
                if (trains[pos].time_departure[ch] == '.') {
                    trains[pos].time_departure[ch] = ':';
                }
                end_ch = ch + 1;
                break;
            }
        }

        minutes_str[0] = trains[pos].time_departure[end_ch]; // Запись минут записи
        minutes_str[1] = trains[pos].time_departure[end_ch + 1];

        int hours = std::stoi(hours_str); // Перевод в целое число
        int mins = std::stoi(minutes_str);

        if (hours < 0 || hours >= 24 || mins < 0 || mins >= 60) { // Проверка условий
            std::cout << "Неправильно задано время!!!" << std::endl;
            is_false = true;
        }
        else {
            is_false = false;
        }
    }

    is_false = true;
    while (is_false) {
        std::cout << "Время в пути: ";
        std::cin.getline(trains[pos].time_way, 7);

        char hours_str[3] = " ", minutes_str[3] = " "; // Промежуточные переменные для часов
            и минут
        int end_ch;

        for (int ch = 0; ch < strlen(trains[pos].time_way); ch++) { // Запись часов записи
            if (trains[pos].time_way[ch] != ':' && trains[pos].time_way[ch] != '.') { //
                Сравнение с двоеточием
                hours_str[ch] = trains[pos].time_way[ch];
            }
            else { // Если равны, то записываем индекс
                if (trains[pos].time_way[ch] == '.') {
                    trains[pos].time_way[ch] = ':';
                }
                end_ch = ch + 1;
                break;
            }
        }

        minutes_str[0] = trains[pos].time_way[end_ch]; // Запись минут записи
        minutes_str[1] = trains[pos].time_way[end_ch + 1];
```

```
        int hours = std::stoi(hours_str); // Перевод в целое число
        int mins = std::stoi(minutes_str);

        if (hours < 0 || mins < 0 || mins >= 60) { // Проверка условий
            std::cout << "Неправильно задано время!!!" << std::endl;
            is_false = true;
        }
        else {
            is_false = false;
        }
    }

    is_false = true;
    while (true) { // Проверка на правильность ввода
        std::cout << "Количество остановок: ";
        std::cin >> trains[pos].stop_count;

        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(32767, '\n');
            std::cout << "Неправильный ввод!!!" << std::endl;
        }
        else {
            std::cin.ignore(32767, '\n');
            if (trains[pos].stop_count <= 0) {
                std::cout << "Количество остановок должно быть больше нуля!!!" << std::endl;
            }
            else {
                break;
            }
        }
    }

    count++; // Увеличение количества поездов на 1
    create_index_file(trains, count);
}


// Функция изменения полей записи
// ========================================================================================
void change_train(Train* &trains, TrainBuffer* trains_buffer, int& count, int&
buffer_count)
{
    char _number[5]; // Номер записи для изменения
    bool is_stock = false; // Флаг наличия номера
    int _index = 0; // Индекс найденного элемента

    save_trains_in_buffer(trains, trains_buffer, count, buffer_count);

    while (!is_stock) { // Проверка на правильность ввода
        print_train(trains, trains_buffer, count, buffer_count);
        std::cout << "Введите номер поезда для изменения: ";
```

```
    std::cin.getline(_number, 5);

    for (int i = 0; auto train: std::span(trains, count)) { // Цикл по всем записям
        if (strcmp(_number, train.number) == 0) { // Проверка на равенство номеров
            is_stock = true;
            _index = i;
            break;
        }
        i++;
    }
    clear();
    if (!is_stock) {
        std::cout << "Такого номера нет!!!" << std::endl;
    }
}

bool is_working = true; // Флаг работы программы
while (is_working) {
    int action;

    std::cout << " Номер" << '|' << " Конечная станция" << '|' << " Дни следования" <<
        '|' << " Отправление" << '|' << "Время в пути" << '|' << " Кол-во остановок" <<
        '|' << "Время в пути в сутках";

    std::cout << std::endl;
    std::cout.width(7);
    std::cout << trains[_index].number;
    std::cout << '|';
    std::cout.width(20);
    std::cout << trains[_index].end_station;
    std::cout << '|';
    std::cout.width(20);
    std::cout << trains[_index].days;
    std::cout << '|';
    std::cout.width(12);
    std::cout << trains[_index].time_departure;
    std::cout << '|';
    std::cout.width(12);
    std::cout << trains[_index].time_way;
    std::cout << '|';
    std::cout.width(17);
    std::cout << trains[_index].stop_count;

    char hours_str1[3] = " ", minutes_str1[3] = " "; // Промежуточные переменные для
        часов и минут
    int end_ch;

    double time_in_hours;

    for (int ch = 0; ch < strlen(trains[_index].time_way); ch++) { // Запись часов записи
        if (trains[_index].time_way[ch] != ':') { // Сравнение с двоеточием
            hours_str1[ch] = trains[_index].time_way[ch];
        }
```

```
    else { // Если равны, то записываем индекс
        end_ch = ch + 1;
        break;
    }
}

minutes_str1[0] = trains[_index].time_way[end_ch]; // Запись минут записи
minutes_str1[1] = trains[_index].time_way[end_ch + 1];
time_in_hours = float(std::stoi(hours_str1)) + float(std::stoi(minutes_str1)) / 60.0;
    // Подсчёт времени записи

std::cout << '|';
std::cout.width(21);
std::cout << time_in_hours / 24.;
std::cout << std::endl;

print_change_menu();
while (true) {
    std::cout << "Выберите действие: ";
    std::cin >> action;

    if (std::cin.fail()) {
        std::cin.clear();
        std::cin.ignore(32767, '\n');
        std::cout << "Неправильный ввод!!!" << std::endl;
    }
    else {
        std::cin.ignore(32767, '\n');
        break;
    }
}

clear();

bool is_false = true;
switch (action) {
    case 1: // Изменение номера
        is_false = true;
        while (is_false) { // Проверка на правильность ввода
            std::cout << "Номер поезда:";
            char input_buffer[1024];
            std::cin.getline(input_buffer, 1024);

            is_false = false;
            if (strlen(input_buffer) > 4) {
                std::cout << "Номер поезда должен состоять из менее, чем 5 символов!!!"
                    << std::endl;
                is_false = true;
                continue;
            }
            else {
                for (int i = 0; i < strlen(input_buffer); i++) {
                    if (int(input_buffer[i]) < 48 || int(input_buffer[i]) > 57) {
```

```
                is_false = true;
                std::cout << "Номер должен состоять из цифр!!!" << std::endl;
                break;
            }
        }
        if (is_false) {
            continue;
        }
        strcpy(trains[_index].number, input_buffer);
    }
    for (int i = 0; auto train: std::span(trains, count)) { // Цикл по всем
        записям
        if (i == _index) continue;
        if (strcmp(trains[_index].number, train.number) == 0) { // Проверка на
            равенство номеров
            is_false = true;
            std::cout << "Такой номер уже есть в базе данных!!!" << std::endl;
        }
        i++;
    }
}
break;
case 2: // Изменение конечной станции
    std::cout << "введите новое название конечной станции: ";
    std::cin.getline(trains[_index].end_station, 256);
    break;
case 3: // Изменение дней следования
    std::cout << "Введите новые дни следования: ";
    std::cin.getline(trains[_index].days, 64);
    break;
case 4: // Изменение времени отправления
    is_false = true;
    while (is_false) {
        std::cout << "Введите новое время отправления: ";
        std::cin.getline(trains[_index].time_departure, 7);

        char hours_str[3] = " ", minutes_str[3] = " "; // Промежуточные переменные
            для часов и минут
        int end_ch;

        for (int ch = 0; ch < strlen(trains[_index].time_departure); ch++) { //
            Запись часов записи
            if       (trains[_index].time_departure[ch]       !=       ':'       &&
                trains[_index].time_departure[ch]   !=   '.')   {   //   Сравнение   с
                двоеточием
                hours_str[ch] = trains[_index].time_departure[ch];
            }
            else { // Если равны, то записываем индекс
                if (trains[_index].time_departure[ch] == '.') {
                    trains[_index].time_departure[ch] = ':';
                }
                end_ch = ch + 1;
                break;
```

```
            }
        }

        minutes_str[0] = trains[_index].time_departure[end_ch]; // Запись минут
            записи
        minutes_str[1] = trains[_index].time_departure[end_ch + 1];

        int hours = std::stoi(hours_str); // Перевод в целое число
        int mins = std::stoi(minutes_str);

        if (hours < 0 || mins < 0 || mins >= 60) { // Проверка условий
            std::cout << "Неправильно задано время!!!" << std::endl;
            is_false = true;
        }
        else {
            is_false = false;
        }
    }
    break;
case 5: // Изменение времени в пути
    is_false = true;
    while (is_false) {
        std::cout << "Введите новое время в пути: ";
        std::cin.getline(trains[_index].time_way, 7);

        char hours_str[3] = " ", minutes_str[3] = " "; // Промежуточные переменные
            для часов и минут
        int end_ch;

        for (int ch = 0; ch < strlen(trains[_index].time_way); ch++) { // Запись
            часов записи
            if (trains[_index].time_way[ch] != ':' && trains[_index].time_way[ch] !
                = '.') { // Сравнение с двоеточием
                hours_str[ch] = trains[_index].time_way[ch];
            }
            else { // Если равны, то записываем индекс
                if (trains[_index].time_way[ch] == '.') {
                    trains[_index].time_way[ch] = ':';
                }
                end_ch = ch + 1;
                break;
            }
        }

        minutes_str[0] = trains[_index].time_way[end_ch]; // Запись минут записи
        minutes_str[1] = trains[_index].time_way[end_ch + 1];

        int hours = std::stoi(hours_str); // Перевод в целое число
        int mins = std::stoi(minutes_str);

        if (hours < 0 || mins < 0 || mins >= 60) { // Проверка условий
            std::cout << "Неправильно задано время!!!" << std::endl;
            is_false = true;
```

```
                }
                else {
                    is_false = false;
                }
            }
            break;
        case 6: // Изменение количества остановок
            while (true) {
                std::cout << "Введите новое количество остановок: ";
                std::cin >> trains[_index].stop_count;

                if (std::cin.fail()) {
                    std::cin.clear();
                    std::cin.ignore(32767, '\n');
                    std::cout << "Неправильный ввод!!!" << std::endl;
                }
                else {
                    std::cin.ignore(32767, '\n');
                    if (trains[_index].stop_count <= 0) { // Проверка условия
                        std::cout << "Количество остановок не может быть отрицательным или
                            равным нулю!!!" << std::endl;
                    }
                    else {
                        break;
                    }
                }
            }
            break;
        case 0: // Завершить изменение
            is_working = false;
            break;
        default:
            std::cout << "Такого действия нет" << std::endl;
            break;
      }
   }
   create_index_file(trains, count);
}


// Удаление поезда по номеру в таблице
// ============================================================================
void   delete_train(Train*   &trains,   TrainBuffer*   trains_buffer,   int&   count,   int&
buffer_count)
{
   print_train(trains, trains_buffer, count, buffer_count);
   print_delete_menu();

   int type; // Переменная типа удаления

   while (true) {
      std::cout << "Выберите тип удаления: ";
      std::cin >> type;
```

```
    if (std::cin.fail()) {
        std::cin.clear();
        std::cin.ignore(32767, '\n');
        std::cout << "Неправильный ввод!!!" << std::endl;
    }
    else {
        std::cin.ignore(32767, '\n');
        if (type < 0 || type > 5) {
            std::cout << "Такого типа нет!!!" << std::endl;
        }
        else {
            break;
        }
    }
}

switch (type) { // Выбор по типу удаления
    case 1: // Удаление по номеру
        char number[5];
        std::cout << "Введите номер: ";
        std::cin.getline(number, 5);
        save_trains_in_buffer(trains, trains_buffer, count, buffer_count);
        delete_train_by_number(trains, count, number);
        break;
    case 2: // Удаление по конечной станции
        char end_station[256];
        std::cout << "Введите название станции: ";
        std::cin.getline(end_station, 256);
        save_trains_in_buffer(trains, trains_buffer, count, buffer_count);
        delete_train_by_end_station(trains, count, end_station);
        break;
    case 3: // Удаление по времени отправления
        char departure_time[7];
        std::cout << "Введите время: ";
        std::cin.getline(departure_time, 7);
        save_trains_in_buffer(trains, trains_buffer, count, buffer_count);
        delete_train_by_departure_time(trains, count, departure_time);
        break;
    case 4: // Удаление по времени в пути
        char way_time[7];
        std::cout << "Введите время: ";
        std::cin.getline(way_time, 7);
        save_trains_in_buffer(trains, trains_buffer, count, buffer_count);
        delete_train_by_way_time(trains, count, way_time);
        break;
    case 5: // Удаление по количеству остановок
        int stop_count;
        while (true) { // Проверка на правильность ввода
            std::cout << "Введите количество остановок: ";
            std::cin >> stop_count;

            if (std::cin.fail()) {
```

```
                std::cin.clear();
                std::cin.ignore(32767, '\n');
                std::cout << "Неправильный ввод!!!" << std::endl;
            }
            else {
                std::cin.ignore(32767, '\n');
                if (stop_count <= 0) {
                    std::cout << "Количество остановок должно быть больше нуля!!!" <<
                        std::endl;
                }
                else {
                    break;
                }
            }
        }
        save_trains_in_buffer(trains, trains_buffer, count, buffer_count);
        delete_train_by_stop_count(trains, count, stop_count);
        break;
    case 0: // Отмена
        break;
    }
    create_index_file(trains, count);
}


// Сортировка поездов
// ============================================================================================
void sort_train(Train*& trains, TrainBuffer* trains_buffer, int& count, int& buffer_count)
{
    int type; // Переменная типа сортировки
    bool reverse; // Флаг направления сортировки (прямая / обратная)
    bool in_file; // Флаг записи в файл

    print_sort_menu(); // Вывод меню типов сортировки
    while (true) {
        std::cout << "Выберите тип сортировки: ";
        std::cin >> type;

        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(32767, '\n');
            std::cout << "Неправильный ввод!!!" << std::endl;
        }
        else {
            std::cin.ignore(32767, '\n');
            if (type < 0 || type > 5) {
                std::cout << "Такого типа нет!!!" << std::endl;
            }
            else {
                break;
            }
        }
    }
```

```
print_reverse_menu(); // Вывод меню флагов сортировки
while (true) {
    std::cout << "Выберите порядок сортировки: ";
    std::cin >> reverse;

    if (std::cin.fail()) {
        std::cin.clear();
        std::cin.ignore(32767, '\n');
        std::cout << "Неправильный ввод!!!" << std::endl;
    }
    else {
        std::cin.ignore(32767, '\n');
        break;
    }
}
print_file_menu(); // Вывод меню флагов записи в файл
while (true) {
    std::cout << "Выберите тип записи в файл: ";
    std::cin >> in_file;

    if (std::cin.fail()) {
        std::cin.clear();
        std::cin.ignore(32767, '\n');
        std::cout << "Неправильный ввод!!!" << std::endl;
    }
    else {
        std::cin.ignore(32767, '\n');
        break;
    }
}

Train* sort_trains = new Train[count]; // Создание временного массива поездов
for (int i = 0; auto train: std::span(trains, count)) { // Переписываем все поезда из
    оригинального массива
    sort_trains[i] = train;
    i++;
}
std::ifstream file;
int index[count];
bool is_end = false;
switch (type) { // Выбор по типу сортировки
    case 1: // Сортировка по номеру
        file.open("index_file/number_sort.txt", std::ios::binary);
        break;
    case 2: // Сортировка по названию станции
        file.open("index_file/end_station_sort.txt", std::ios::binary);
        break;
    case 3: // Сортировка по времени отправления
        file.open("index_file/departure_time_sort.txt", std::ios::binary);
        break;
    case 4: // Сортировка по времени в пути
        file.open("index_file/way_time_sort.txt", std::ios::binary);
```

```
            break;
        case 5: // Сортировка по количеству остановок
            file.open("index_file/stop_count_sort.txt", std::ios::binary);
            break;
        case 0: // Отмена
            is_end = true;
            break;
    }
    if (!is_end) {
        // Чтение индексного файла
        file.read((char*)&count, sizeof(int));
        for (auto & i: std::span(index, count)) {
            file.read((char*)(&i), sizeof(int));
        }

        // Сортировка в срртветствии с индексами
        int index_index = (reverse) ? Count - 1 : 0;
        for (int i = 0; i < count; i++) {
            sort_trains[i] = trains[index[index_index]];
            if (reverse) {
                index_index--;
            }
            else {
                index_index++;
            }
        }

        print_train(sort_trains, trains_buffer, count, buffer_count); // Вывод поездов на
            экран

        if (in_file) {
            save_trains_in_buffer(trains, trains_buffer, count, buffer_count);
            for (int i = 0; auto train: std::span(sort_trains, count)) {
                trains[i] = train;
                i++;
            }
            create_index_file(trains, count);
        }
    }

    delete[] sort_trains; // Очищение выделенной памяти
}


// Функция выборки записей по полям
// ========================================================================================
void   select_train(Train*&   trains,   TrainBuffer*   trains_buffer,   int&   count,   int&
buffer_count)
{
    int type; // Тип выборки/поиска
    print_train(trains, trains_buffer, count, buffer_count);
    print_selection_menu();
    while (true) {
```

```
    std::cout << "Выберите тип выборки: ";
    std::cin >> type;

    if (std::cin.fail()) {
        std::cin.clear();
        std::cin.ignore(32767, '\n');
        std::cout << "Неправильный ввод!!!" << std::endl;
    }
    else {
        std::cin.ignore(32767, '\n');
        if (type < 0 || type > 5) {
            std::cout << "Такого типа нет!!!" << std::endl;
        }
        else {
            break;
        }
    }
}

switch (type) {
    case 1: // Выборка по номеру
        int down_number, up_number; // Диапазон номеров
        std::cout << "Введите диапазон значений" << std::endl;

        while (true) { // Проверка на правильность ввода
            std::cout << "Нижняя граница: ";
            std::cin >> down_number;

            if (std::cin.fail()) {
                std::cin.clear();
                std::cin.ignore(32767, '\n');
                std::cout << "Неправильный ввод!!!" << std::endl;
            }
            else {
                std::cin.ignore(32767, '\n');
                if (down_number > 9999 || down_number <= 0) {
                    std::cout << "Номер поезда должен состоять из менее, чем 5 символов, и
                        быть больше нуля!!!" << std::endl;
                }
                else {
                    break;
                }
            }
        }
        while (true) { // Проверка на правильность ввода
            std::cout << "Верхняя граница: ";
            std::cin >> up_number;

            if (std::cin.fail()) {
                std::cin.clear();
                std::cin.ignore(32767, '\n');
                std::cout << "Неправильный ввод!!!" << std::endl;
            }
```

```
            else {
                std::cin.ignore(32767, '\n');
                if (up_number > 9999 || up_number <= 0) {
                std::cout << "Номер поезда должен состоять из менее, чем 5 символов, и быть
                    больше нуля!!!" << std::endl;
            }
            else {
                break;
            }
        }
    }
    select_by_number(trains, trains_buffer, count, buffer_count, down_number, up_number);
    break;
case 2: // Выборка по названию конечной станции
    char end_station[256];
    std::cout << "Введите название конечной станции: ";
    std::cin.getline(end_station, 256);
    select_by_end_station(trains, trains_buffer, count, buffer_count, end_station);
    break;
case 3: // Выборка по времени отправления
    char down_departure_time[7], up_departure_time[7]; // Диапазон веремени отправления
    std::cout << "Введите диапазон значений" << std::endl;
    std::cout << "Нижняя граница: ";
    std::cin.getline(down_departure_time, 7);
    std::cout << "Верхняя граница: ";
    std::cin.getline(up_departure_time, 7);
    select_by_departure_time(trains,      trains_buffer,      count,      buffer_count,
        down_departure_time, up_departure_time);
    break;
case 4: // Выборка по времени в пути
    char down_way_time[7], up_way_time[7]; // Диапазон времени в пути
    std::cout << "Введите диапазон значений" << std::endl;
    std::cout << "Нижняя граница: ";
    std::cin.getline(down_way_time, 7);
    std::cout << "Верхняя граница: ";
    std::cin.getline(up_way_time, 7);
    select_by_way_time(trains,   trains_buffer,   count,   buffer_count,   down_way_time,
        up_way_time);
    break;
case 5: // Выборка по количеству остановок
    int down_stop_number, up_stop_number; // Диапазон количества остановок
    std::cout << "Введите диапазон значений" << std::endl;
    while (true) { // Проверка на правильность ввода
        std::cout << "Нижняя граница: ";
        std::cin >> down_stop_number;

        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(32767, '\n');
            std::cout << "Неправильный ввод!!!" << std::endl;
        }
        else {
            std::cin.ignore(32767, '\n');
```

```
            if (down_stop_number <= 0) {
                std::cout << "Количество остановок должно быть больше нуля!!!" <<
                    std::endl;
            }
            else {
                break;
            }
        }
    }
    while (true) { // Проверка на правильность ввода
        std::cout << "Верхняя граница: ";
        std::cin >> up_stop_number;

        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(32767, '\n');
            std::cout << "Неправильный ввод!!!" << std::endl;
        }
        else {
            std::cin.ignore(32767, '\n');
            if (up_stop_number <= 0) {
                std::cout << "Количество остановок должно быть больше нуля!!!" <<
                    std::endl;
            }
            else {
                break;
            }
        }
    }
    select_by_stop_count(trains, trains_buffer, count, buffer_count, down_stop_number,
        up_stop_number);
    break;
case 0: // Отмена
    break;
}
}


// Функция отмены последнего действия
// ============================================================================================
void undo_action(Train* &trains, TrainBuffer* trains_buffer, int& count, int& buffer_count)
{
    if (buffer_count != 0) { // Проверка на наличие состояний для отмены
        count = trains_buffer[buffer_count - 1].count; // Получение количества поездов
        delete[] trains; // Удаление указателя
        trains = new Train[count]; // Создание нового с новым количеством
        for (int i = 0; auto train: std::span(trains_buffer[buffer_count - 1].trains, count))
            {
            trains[i] = train; // Перепись данных из буфера в массив
            i++;
        }
        delete[] trains_buffer[buffer_count - 1].trains; // Удаление буфера
        buffer_count--; // Уменьшение количества сохраненных состояний
```

```
        create_index_file(trains, count);
    }
}
```

menu.h

```
#ifndef MENU_H
#define MENU_H


void print_menu();
void print_sort_menu();
void print_reverse_menu();
void print_file_menu();
void print_delete_menu();
void print_selection_menu();
void print_change_menu();

#endif
```

menu.cpp

```
/*
Файл с функциями меню (вывод меню для различных действий)
*/

#include "menu.h"

#include <iostream>


// Вывод меню действий
// ========================================================================================
void print_menu()
{
    std::cout << std::endl;
    std::cout << "====================MENU====================" << std::endl;
    std::cout << std::endl;
    std::cout
    << "1" << '\t' << "вывод всех поездов" << std::endl
    << "2" << '\t' << "добавление нового поезда" << std::endl
    << "3" << '\t' << "изменение поезда" << std::endl
    << "4" << '\t' << "удаление поезда" << std::endl
    << "5" << '\t' << "сортировка поездов" << std::endl
    << "6" << '\t' << "выборка записей" << std::endl
    << "7" << '\t' << "отмена последнего действия" << std::endl
    << "0" << '\t' << "выход из программы" << std::endl;
    std::cout << std::endl;
}


// Вывод меню типов сортировки
// ========================================================================================
void print_sort_menu()
{
```

```
    std::cout << std::endl;
    std::cout << "==================SORT MENU==================" << std::endl;
    std::cout << std::endl;
    std::cout
    << "1" << '\t' << "сортировка по номеру" << std::endl
    << "2" << '\t' << "сортировка по конечной станции" << std::endl
    << "3" << '\t' << "сортировка по времени отправления" << std::endl
    << "4" << '\t' << "сортировка по времени в пути" << std::endl
    << "5" << '\t' << "сортировка по количеству остановок" << std::endl
    << "0" << '\t' << "отмена" << std::endl;
    std::cout << std::endl;
}


// Вывод меню флагов сортировки
// ============================================================================================
void print_reverse_menu()
{
    std::cout << std::endl;
    std::cout << "==================REVERSE==================" << std::endl;
    std::cout << std::endl;
    std::cout
    << "0" << '\t' << "прямая сортировка (от меньшего к большему)" << std::endl
    << "1" << '\t' << "обратная сортировка (от большего к меньшему)" << std::endl;
    std::cout << std::endl;
}


// Вывод меню флагов записи в файл
// ============================================================================================
void print_file_menu()
{
    std::cout << std::endl;
    std::cout << "====================FILE====================" << std::endl;
    std::cout << std::endl;
    std::cout
    << "0" << '\t' << "без записи в файл" << std::endl
    << "1" << '\t' << "с записью в файл" << std::endl;
    std::cout << std::endl;
}


// Вывод меню типов удаления
// ============================================================================================
void print_delete_menu()
{
    std::cout << std::endl;
    std::cout << "====================DELETE====================" << std::endl;
    std::cout << std::endl;
    std::cout
    << "1" << '\t' << "удаление записи по номеру" << std::endl
    << "2" << '\t' << "удаление записей по названию станции" << std::endl
    << "3" << '\t' << "удаление записей по времени отправления" << std::endl
```

```
    << "4" << '\t' << "удаление записей по времени в пути" << std::endl
    << "5" << '\t' << "удаление записей по количеству остановок" << std::endl
    << "0" << '\t' << "отмена" << std::endl;
    std::cout << std::endl;
}


// Вывод меню типов выборки
// ============================================================================================
void print_selection_menu()
{
    std::cout << std::endl;
    std::cout << "====================SELECT====================" << std::endl;
    std::cout << std::endl;
    std::cout
    << "1" << '\t' << "выборка записей по номеру" << std::endl
    << "2" << '\t' << "выборка записей по названию станции" << std::endl
    << "3" << '\t' << "выборка записей по времени отправления" << std::endl
    << "4" << '\t' << "выборка записей по времени в пути" << std::endl
    << "5" << '\t' << "выборка записей по количеству остановок" << std::endl
    << "0" << '\t' << "отмена" << std::endl;
    std::cout << std::endl;
}


// Вывод меню изменения записи
// ============================================================================================
void print_change_menu()
{
    std::cout << std::endl;
    std::cout << "====================CHANGE====================" << std::endl;
    std::cout << std::endl;
    std::cout
    << "1" << '\t' << "изменить номер" << std::endl
    << "2" << '\t' << "изменить название станции" << std::endl
    << "3" << '\t' << "изменить дни следования" << std::endl
    << "4" << '\t' << "изменить время отправления" << std::endl
    << "5" << '\t' << "изменить время в пути" << std::endl
    << "6" << '\t' << "изменить количество остановок" << std::endl
    << "0" << '\t' << "завершить изменение" << std::endl;
    std::cout << std::endl;
}
```

file.h

```
#ifndef FILE_H
#define FILE_H

#include "train.h"


int get_records_from_file(char[], Train*&, int&);
int put_records_in_file(char[], Train*, int);
void create_index_file(Train*, int);
```

```
#endif
```

file.cpp

```
/*
Файл с функциями для работы с файлами (чтение, запись)
*/

#include "file.h"
#include "config_func.h"
#include "sort_func.h"

#include <fstream>
#include <iostream>
#include <cstring>
#include <iomanip>
#include <span>


// Получение записей из файла
// ============================================================================================
int get_records_from_file(char file_name[], Train* &trains, int& count)
{
    std::ifstream file(file_name, std::ios::binary); // Создание потока чтения из файла

    if (!file) { // Проверка на ошибки при открытии файла
        std::cout << "Файл не открыт!!!" << std::endl;
        return 1;
    }

    int train_count_in_file;
    file.read((char*)(&train_count_in_file), sizeof(train_count_in_file));

    for (int i = 0; i < train_count_in_file; i++) { // Если файл открыт, то считываем данные
        add_memory_train(trains, count);
        file.read((char*)(&trains[count]), sizeof(Train));
        count++;
    }

    return 0;
}


// Запись данных в файл
// ============================================================================================
int put_records_in_file(char file_name[], Train* trains, int count)
{
    std::ofstream file(file_name, std::ios::binary); // Создание потока записи в файл
    std::ofstream text_file("output_file.txt");

    if (!file || !text_file) {
        std::cout << "Файл не открыт!!!" << std::endl;
```

```
        return 1;
    }

    file.write((char*)(&count), sizeof(count)); // Запись количества записей в файл

    text_file << " Номер" << '|' << " Конечная станция" << '|' << " Дни следования" << '|'
    << " Отправление" << '|' << "Время в пути" << '|' << " Кол-во остановок" << '|' <<
        "Время в пути в сутках";
    text_file << std::endl; // Запись шапки в текстовый файл

    for (auto & train: std::span(trains, count)) { // Цикл по количеству записей
        file.write((char*)(&train), sizeof(Train)); // Запись структуры в файл

        // Запись полей структуры в текстовый файл
        text_file << std::setw(7) << train.number << '|';
        text_file << std::setw(20) << train.end_station << '|';
        text_file << std::setw(20) << train.days << '|';
        text_file << std::setw(12) << train.time_departure << '|';
        text_file << std::setw(12) << train.time_way << '|';
        text_file << std::setw(17) << train.stop_count << '|';

        // Расчет поля время в пути в сутках
        char hours_str1[3] = " ", minutes_str1[3] = " "; // Промежуточные переменные для
            часов и минут
        int end_ch;
        double time_in_hours;

        for (int ch = 0; ch < strlen(train.time_way); ch++) { // Запись часов записи
            if (train.time_way[ch] != ':') { // Сравнение с двоеточием
                hours_str1[ch] = train.time_way[ch];
            }
            else { // Если равны, то записываем индекс
                end_ch = ch + 1;
                break;
            }
        }

        minutes_str1[0] = train.time_way[end_ch]; // Запись минут записи
        minutes_str1[1] = train.time_way[end_ch + 1];
        time_in_hours = float(std::stoi(hours_str1)) + float(std::stoi(minutes_str1)) / 60.0;
            // Подсчёт времени записи

        text_file << std::setw(21) << time_in_hours / 24.;
        text_file << std::endl;
    }
    return 0;
}


// Функция создания индексных файлов
// ========================================================================================
void create_index_file(Train* trains, int count)
{
```

```
Train* trains_buffer = new Train[count];
int* index = new int[count];
for (int i = 0; i < count; i++) {
   trains_buffer[i] = trains[i];
}
number_sort(trains_buffer, index, count);
std::ofstream file("index_file/number_sort.txt", std::ios::binary);

file.write((char*)(&count), sizeof(int));
for (int i = 0; i < count; i++) {
   file.write((char*)(&index[i]), sizeof(int));
}
file.close();

for (int i = 0; i < count; i++) {
   trains_buffer[i] = trains[i];
}
end_station_sort(trains_buffer, index, count);
file.open("index_file/end_station_sort.txt", std::ios::binary);

file.write((char*)(&count), sizeof(int));
for (int i = 0; i < count; i++) {
   file.write((char*)(&index[i]), sizeof(int));
}
file.close();

for (int i = 0; i < count; i++) {
   trains_buffer[i] = trains[i];
}
departure_time_sort(trains_buffer, index, count);
file.open("index_file/departure_time_sort.txt", std::ios::binary);

file.write((char*)(&count), sizeof(int));
for (int i = 0; i < count; i++) {
   file.write((char*)(&index[i]), sizeof(int));
}
file.close();

for (int i = 0; i < count; i++) {
   trains_buffer[i] = trains[i];
}
way_time_sort(trains_buffer, index, count);
file.open("index_file/way_time_sort.txt", std::ios::binary);

file.write((char*)(&count), sizeof(int));
for (int i = 0; i < count; i++) {
   file.write((char*)(&index[i]), sizeof(int));
}
file.close();

for (int i = 0; i < count; i++) {
   trains_buffer[i] = trains[i];
}
```

```
    stop_count_sort(trains_buffer, index, count);
    file.open("index_file/stop_count_sort.txt", std::ios::binary);

    file.write((char*)(&count), sizeof(int));
    for (int i = 0; i < count; i++) {
        file.write((char*)(&index[i]), sizeof(int));
    }
    file.close();
}
```

clear.h

```
#ifndef CLEAR_H
#define CLEAR_H

void clear();

#endif
```

clear.cpp

```
#include "clear.h"

#include <iostream>


// Функция очистки экрана для linux
// ================================================================================
void clear()
{
    system("clear");
}
```

config_func.h

```
#ifndef CONFIG_FUNC_H
#define CONFIG_FUNC_H

#include "train.h"


void add_memory_train(Train*&, int&);
void sub_memory_train(Train*&, int&);
void save_trains_in_buffer(Train*, TrainBuffer*, int, int&);

#endif
```

config_func.cpp

```
/*
Файл с функциями для работы с памятью
*/

#include "config_func.h"

#include <span>
```

```
// Функция выделения дополнительной ячейки памяти
// ============================================================================================
void add_memory_train(Train* &trains, int& count)
{
    Train* buffer_trains = new Train[count]; // Создание временного буфера

    for (int i = 0; auto train: std::span(trains, count)) { // Перепись данных в буфер
        buffer_trains[i] = train;
        i++;
    }

    delete[] trains; // Удаление указателя
    trains = new Train[count + 1]; // Создание нового указателя с большим количеством
        элементов

    for (int i = 0; auto train: std::span(buffer_trains, count)) { // Перепись данных из
        буфера в массив
        trains[i] = train;
        i++;
    }
    delete[] buffer_trains; // Удаление буфера
}


// Функция очищения ячейки памяти
// ============================================================================================
void sub_memory_train(Train* &trains, int& count)
{
    Train* buffer_trains = new Train[count - 1]; // Создание временного буфера

    for (int i = 0; i < count - 1; i++) { // Перепись данных в буфер
        buffer_trains[i] = trains[i];
    }

    delete[] trains; // Удаление указателя
    trains = new Train[count - 1]; // Создание нового указателя с меньшим количеством
        элементов

    for (int i = 0; i < count - 1; i++) { // Перепись данных из буфера в массив
        trains[i] = buffer_trains[i];
    }
    delete[] buffer_trains; // Удаление буфера
}


// Функция сохраниения состояния массива в буфер для отмены
// ============================================================================================
void save_trains_in_buffer(Train* trains, TrainBuffer* trains_buffer, int count, int&
buffer_count)
{
    if (buffer_count == 10) {
```

```
        delete[] trains_buffer[0].trains; // Удаление первого состояния
        for (int i = 0; i < buffer_count - 1; i++) { // Сдвиг всех элементов назад на один
            trains_buffer[i] = trains_buffer[i + 1];
        }
        buffer_count = 9; // Изменение количества
    }

    trains_buffer[buffer_count].count = count; // Запись количества поездов
    trains_buffer[buffer_count].trains = new Train[count]; // Запись поездов
    for (int i = 0; auto train: std::span(trains, count)) { // Перепись поездов из массива в
        буфер для отмены
        trains_buffer[buffer_count].trains[i] = train;
        i++;
    }

    buffer_count++; // Увеличение количества состояний в буфере
}


/*
Файл с функциями для удаления записей
*/

#include "delete_func.h"
#include "config_func.h"
#include "file.h"

#include <iostream>
#include <cstring>
```

delete_func.h

```
#ifndef DELETE_FUNC_H

#define DELETE_FUNC_H

#include "train.h"


void delete_train_by_number(Train*&, int&, char[5]);
void delete_train_by_end_station(Train*&, int&, char[256]);
void delete_train_by_departure_time(Train*&, int&, char[7]);
void delete_train_by_way_time(Train*&, int&, char[7]);
void delete_train_by_stop_count(Train*&, int&, int);

#endif
```

delete_func.cpp

```
// Удаление поезда по номеру
```

```
// ============================================================================================
void delete_train_by_number(Train* &trains, int& count, char number[5])
{
    int i = 0;
    while (i < count) {
        if (strcmp(trains[i].number, number) == 0) { // Сравнение номеров
            if (i != count - 1) {
                for (int j = i; j < count - 1; j++) { // Сдвиг записей на один назад
                    trains[j] = trains[j + 1];
                }

                sub_memory_train(trains, count);
                count--;
                break;
            }
            else {
                sub_memory_train(trains, count);
                count--;
                break;
            }
        }
        i++;
    }
    create_index_file(trains, count);
}


// Удаление поездов по названию конечной станции
// ============================================================================================
void delete_train_by_end_station(Train* &trains, int& count, char end_station[256])
{
    int i = 0;
    while (i < count) {
        bool is_equal = true;
        if (strlen(trains[i].end_station) < strlen(end_station)) { // Проверка на длину
            is_equal = false;
        }
        else {
            for (int j = 0; j < strlen(end_station); j++) { // Сравнение названия станции
                if (trains[i].end_station[j] != end_station[j]) {
                    is_equal = false;
                    break;
                }
            }
        }
        if (is_equal) {
            for (int j = i; j < count - 1; j++) { // Сдвиг на один назад
                trains[j] = trains[j + 1];
            }
            i--;
            sub_memory_train(trains, count);
            count--;
        }
```

```
        i++;
    }
    create_index_file(trains, count);
}


// Удаление поездов по времени отправления
// ============================================================================================
void delete_train_by_departure_time(Train* &trains, int& count, char departure_time[7])
{
    int i = 0;
    while (i < count) {
        bool is_equal = true;
        if (strlen(trains[i].time_departure) < strlen(departure_time)) { // Проверка на длину
            is_equal = false;
        }
        else {
            for (int j = 0; j < strlen(departure_time); j++) { // Сравнение времени
                отправления
                if (trains[i].time_departure[j] != departure_time[j]) {
                    is_equal = false;
                    break;
                }
            }
        }
        if (is_equal) {
            for (int j = i; j < count - 1; j++) { // Сдвиг на один назад
                trains[j] = trains[j + 1];
            }
            i--;
            sub_memory_train(trains, count);
            count--;
        }
        i++;
    }
    create_index_file(trains, count);
}


// Удаление поездов по времени в пути
// ============================================================================================
void delete_train_by_way_time(Train* &trains, int& count, char way_time[7])
{
    int i = 0;
    while (i < count) {
        bool is_equal = true;
        if (strlen(trains[i].time_way) < strlen(way_time)) { // Проверка на длину
            is_equal = false;
        }
        else {
            for (int j = 0; j < strlen(way_time); j++) { // Сравнение времени в пути
                if (trains[i].time_way[j] != way_time[j]) {
                    is_equal = false;
```

```
                break;
            }
        }
    }
    if (is_equal) {
        for (int j = i; j < count - 1; j++) { // Сдвиг на один назад
            trains[j] = trains[j + 1];
        }
        i--;
        sub_memory_train(trains, count);
        count--;
    }
    i++;
}
create_index_file(trains, count);
}


// Удаление поездов по количеству остановок
// ========================================================================================
void delete_train_by_stop_count(Train* &trains, int& count, int stop_count)
{
    int i = 0;
    while (i < count) {
        if (trains[i].stop_count == stop_count) { // Сравнение количества остановок
            for (int j = i; j < count - 1; j++) { // Сдвиг на один назад
                trains[j] = trains[j + 1];
            }
            i--;
            sub_memory_train(trains, count);
            count--;
        }
        i++;
    }
    create_index_file(trains, count);
}
```

select_func.h

```
#ifndef SELECT_FUNC_H
#define SELECT_FUNC_H

#include "train.h"


void select_by_number(Train*, TrainBuffer*, int, int, int, int);
void select_by_end_station(Train*, TrainBuffer*, int, int, char[256]);
void select_by_departure_time(Train*, TrainBuffer*, int, int, char[7], char[7]);
void select_by_way_time(Train*, TrainBuffer*, int, int, char[7], char[7]);
void select_by_stop_count(Train*, TrainBuffer*, int, int, int, int);

#endif
```

select_func.cpp

```
/*
Файл с функциями выборки записей по полям
*/

#include "select_func.h"
#include "train_func.h"
#include "train.h"

#include <iostream>
#include <cstring>
#include <span>


// Выборка по номеру
// ============================================================================================
void select_by_number(Train* trains, TrainBuffer* trains_buffer1, int count, int
buffer_count, int down_number, int up_number)
{
    Train* trains_buffer = new Train[count]; // Буфер поездов для выборки
    int selection_count = 0; // Количество найденных поездов

    for (int i = 0; auto train: std::span(trains, count)) { // Цикл по записям
        int _number = std::stoi(train.number);
        if (_number >= down_number && _number <= up_number) { // Проверка в диапазоне
            trains_buffer[selection_count] = train;
            selection_count++;
        }
        i++;
    }
    print_train(trains_buffer, trains_buffer1, selection_count, buffer_count);
    std::cout << "Количество записей найдено: " << selection_count << std::endl;
    delete[] trains_buffer; // Удаление буфера
}


// Выборка по названию конечной станции
// ============================================================================================
void select_by_end_station(Train* trains, TrainBuffer* trains_buffer1, int count, int
buffer_count, char end_station[256])
{
    Train* trains_buffer = new Train[count]; // Буфер поездов для выборки
    int selection_count = 0; // Количество найденных поездов

    int i = 0;
    while (i < count) { // Цикл по записям
        bool is_equal = true;
        if (strlen(trains[i].end_station) < strlen(end_station)) { // Проверка на длину
            строки
            is_equal = false;
        }
        else {
            for (int j = 0; j < strlen(end_station); j++) {
```

```
            if (trains[i].end_station[j] != end_station[j]) { // Сравнение символов
                is_equal = false;
                break;
            }
        }
    }
    if (is_equal) {
        trains_buffer[selection_count] = trains[i];
        selection_count++;
    }
    i++;
}
print_train(trains_buffer, trains_buffer1, selection_count, buffer_count);
std::cout << "Количество записей найдено: " << selection_count << std::endl;
delete[] trains_buffer; // Удаление буфера
}


// Выборка по времени отправления
// ========================================================================================
void select_by_departure_time(Train* trains, TrainBuffer* trains_buffer1, int count, int
buffer_count, char down_departure_time[7], char up_departure_time[7])
{
    Train* trains_buffer = new Train[count]; // Буфер поездов для выборки
    int selection_count = 0; // Количество найденных поездов

    float down_time, up_time; // Пременные времени в часах (дробные)
    int end_ch; // Индекс конца цифр часов

    char hours_str2[3] = " ", minutes_str2[3] = " ", hours_str1[3] = " ", minutes_str1[3] =
        " "; // Промежуточные переменные для часов и минут

    for (int ch = 0; ch < strlen(down_departure_time); ch++) {
        if (down_departure_time[ch] != ':') { // Сравнение с двоеточием
            hours_str1[ch] = down_departure_time[ch];
        }
        else { // Если равны, то записываем индекс
            end_ch = ch + 1;
            break;
        }
    }

    minutes_str1[0] = down_departure_time[end_ch]; // Запись минут i-ой записи
    minutes_str1[1] = down_departure_time[end_ch + 1];
    down_time = float(std::stoi(hours_str1)) + float(std::stoi(minutes_str1)) / 60.0; //
        Подсчёт времени i-ой записи

    for (int ch = 0; ch < strlen(up_departure_time); ch++) {
        if (up_departure_time[ch] != ':') {
            hours_str2[ch] = up_departure_time[ch];
        }
        else {
            end_ch = ch + 1;
```

```
            break;
        }
    }
    minutes_str2[0] = up_departure_time[end_ch];
    minutes_str2[1] = up_departure_time[end_ch + 1];
    up_time = float(std::stoi(hours_str2)) + float(std::stoi(minutes_str2)) / 60.0;

    int i = 0;
    while (i < count) {
        bool is_equal = true;

        double _time;
        char _hours_str[3] = " ", _minutes_str[3] = " ";

        for (int ch = 0; ch < strlen(trains[i].time_departure); ch++) {
            if (trains[i].time_departure[ch] != ':') {
                _hours_str[ch] = trains[i].time_departure[ch];
            }
            else {
                end_ch = ch + 1;
                break;
            }
        }
        _minutes_str[0] = trains[i].time_departure[end_ch];
        _minutes_str[1] = trains[i].time_departure[end_ch + 1];
        _time = float(std::stoi(_hours_str)) + float(std::stoi(_minutes_str)) / 60.0;

        if (_time >= down_time && _time <= up_time) {
            trains_buffer[selection_count] = trains[i];
            selection_count++;
        }
        i++;
    }
    print_train(trains_buffer, trains_buffer1, selection_count, buffer_count);
    std::cout << "Количество записей найдено: " << selection_count << std::endl;
    delete[] trains_buffer; // Удаление буфера
}


// Выборка по времени в пути
// ========================================================================================
void select_by_way_time(Train* trains, TrainBuffer* trains_buffer1, int count, int
buffer_count, char down_way_time[7], char up_way_time[7])
{
    Train* trains_buffer = new Train[count]; // Буфер поездов для выборки
    int selection_count = 0; // Количество найденных поездов

    float down_time, up_time; // Пременные времени в часах (дробные)
    int end_ch; // Индекс конца цифр часов

    char hours_str2[3] = " ", minutes_str2[3] = " ", hours_str1[3] = " ", minutes_str1[3] =
        " "; // Промежуточные переменные для часов и минут
```

```
for (int ch = 0; ch < strlen(down_way_time); ch++) {
    if (down_way_time[ch] != ':') { // Сравнение с двоеточием
        hours_str1[ch] = down_way_time[ch];
    }
    else { // Если равны, то записываем индекс
        end_ch = ch + 1;
        break;
    }
}

minutes_str1[0] = down_way_time[end_ch]; // Запись минут i-ой записи
minutes_str1[1] = down_way_time[end_ch + 1];
down_time = float(std::stoi(hours_str1)) + float(std::stoi(minutes_str1)) / 60.0; //
    Подсчёт времени i-ой записи

for (int ch = 0; ch < strlen(up_way_time); ch++) {
    if (up_way_time[ch] != ':') {
        hours_str2[ch] = up_way_time[ch];
    }
    else {
        end_ch = ch + 1;
        break;
    }
}
minutes_str2[0] = up_way_time[end_ch];
minutes_str2[1] = up_way_time[end_ch + 1];
up_time = float(std::stoi(hours_str2)) + float(std::stoi(minutes_str2)) / 60.0;

int i = 0;
while (i < count) {
    bool is_equal = true;

    double _time;
    char _hours_str[3] = " ", _minutes_str[3] = " ";

    for (int ch = 0; ch < strlen(trains[i].time_way); ch++) {
        if (trains[i].time_way[ch] != ':') {
            _hours_str[ch] = trains[i].time_way[ch];
        }
        else {
            end_ch = ch + 1;
            break;
        }
    }
    _minutes_str[0] = trains[i].time_way[end_ch];
    _minutes_str[1] = trains[i].time_way[end_ch + 1];
    _time = float(std::stoi(_hours_str)) + float(std::stoi(_minutes_str)) / 60.0;

    if (_time >= down_time && _time <= up_time) {
        trains_buffer[selection_count] = trains[i];
        selection_count++;
    }
    i++;
```

```
    }
    print_train(trains_buffer, trains_buffer1, selection_count, buffer_count);
    std::cout << "Количество записей найдено: " << selection_count << std::endl;
    delete[] trains_buffer; // Удаление буфера
}


// Выборка по количеству остановок
// ============================================================================
void select_by_stop_count(Train* trains, TrainBuffer* trains_buffer1, int count, int
buffer_count, int down_stop_count, int up_stop_count)
{
    Train* trains_buffer = new Train[count]; // Буфер поездов для выборки
    int selection_count = 0; // Количество найденных поездов

    for (int i = 0; auto train: std::span(trains, count)) { // Цикл по записям
        int _number = train.stop_count;
        if (_number >= down_stop_number && _number <= up_stop_number) { // Проверка на
            диапазон
            trains_buffer[selection_count] = train;
            selection_count++;
        }
        i++;
    }
    print_train(trains_buffer, trains_buffer1, selection_count, buffer_count);
    std::cout << "Количество записей найдено: " << selection_count << std::endl;
    delete[] trains_buffer; // Удаление буфера
}
```

sort_func.h

```
#ifndef SORT_FUNC_H
#define SORT_FUNC_H

#include "train.h"


void number_sort(Train*, int*, int);
void end_station_sort(Train*, int*, int);
void departure_time_sort(Train*, int*, int);
void way_time_sort(Train*, int*, int);
void stop_count_sort(Train*, int*, int);

void quick_sort(int*, int, int, int*);
void quick_sort(double*, int, int, int*);
void quick_sort(char[][256], int, int, int*);

#endif
```

sort_func.cpp

```
/*
Файл с функциями сортировки записей по полям
```

```
*/

#include "sort_func.h"

#include <cstring>
#include <iomanip>
#include <iostream>
#include <span>


// Сортировка по номеру
// ============================================================================================
void number_sort(Train* trains, int* index, int count)
{
    int _number[count];
    for (int i = 0; auto train: std::span(trains, count)) {
        index[i] = i;
        _number[i] = std::stoi(train.number);
        i++;
    }
    quick_sort(_number, 0, count - 1, index);
}


// Сортировка по названию станции
// ============================================================================================
void end_station_sort(Train* trains, int* index, int count)
{
    char _end_station[count][256];
    for (int i = 0; auto train: std::span(trains, count)) {
        char temp[256];
        index[i] = i;
        strcpy(temp, train.end_station);
        for (int j = 0; j < strlen(temp); j++) {
            temp[j] = tolower(temp[j]);
        }
        strcpy(_end_station[i], temp);
        i++;
    }
    quick_sort(_end_station, 0, count - 1, index);
}


// Сортировка по времени отправления
// ============================================================================================
void departure_time_sort(Train* trains, int* index, int count)
{
    double _departure_time[count];
    for (int i = 0; auto train: std::span(trains, count)) {
        int end_ch;
        index[i] = i;
        char hours_str1[3] = " ", minutes_str1[3] = " "; // Промежуточные переменные для
            часов и минут
```

```
        for (int ch = 0; ch < strlen(trains[i].time_departure); ch++) { // Запись часов i-ой
            записи
            if (train.time_departure[ch] != ':') { // Сравнение с двоеточием
                hours_str1[ch] = train.time_departure[ch];
            }
            else { // Если равны, то записываем индекс
                end_ch = ch + 1;
                break;
            }
        }

        minutes_str1[0] = train.time_departure[end_ch]; // Запись минут i-ой записи
        minutes_str1[1] = train.time_departure[end_ch + 1];
        _departure_time[i]            =            double(std::stoi(hours_str1))            +
            double(std::stoi(minutes_str1)) / 60.0; // Подсчёт времени i-ой записи
        i++;
    }

    quick_sort(_departure_time, 0, count - 1, index);
}


// Сортировка по времени в пути
// Отличие только в поле, по которому сортируем
// ============================================================================================
void way_time_sort(Train* trains, int* index, int count)
{
    double _way_time[count];
    for (int i = 0; auto train: std::span(trains, count)) {
        int end_ch;
        index[i] = i;
        char hours_str1[3] = " ", minutes_str1[3] = " "; // Промежуточные переменные для
            часов и минут

        for (int ch = 0; ch < strlen(train.time_way); ch++) { // Запись часов i-ой записи
            if (train.time_way[ch] != ':') { // Сравнение с двоеточием
                hours_str1[ch] = train.time_way[ch];
            }
            else { // Если равны, то записываем индекс
                end_ch = ch + 1;
                break;
            }
        }

        minutes_str1[0] = train.time_way[end_ch]; // Запись минут i-ой записи
        minutes_str1[1] = train.time_way[end_ch + 1];
        _way_time[i]  =  double(std::stoi(hours_str1))  +  double(std::stoi(minutes_str1))  /
            60.0; // Подсчёт времени i-ой записи
        i++;
    }
    quick_sort(_way_time, 0, count - 1, index);
}
```

```
// Сортировка по количеству остановок
// ============================================================================================
void stop_count_sort(Train* trains, int* index, int count)
{
    int _stop_count[count];
    for (int i = 0; auto train: std::span(trains, count)) {
        index[i] = i;
        _stop_count[i] = train.stop_count;
        i++;
    }

    quick_sort(_stop_count, 0, count - 1, index);
}


// Функции для быстрой сортировки
// ============================================================================================
void quick_sort(int* A, int from, int to, int* index)
{
    int x, i, j, temp;

    if (from >= to) return; // Условие окончания рекурсии
    i = from; // Рассматриваем элементы с A[from] до A[to]
    j = to;
    x = A[(from + to) / 2]; // Выбираем средний элемент

    while ( i <= j ) {
        while (A[i] < x) i++; // Ищем пару для перестановки
        while (A[j] > x) j--;
        if (i <= j) {
            temp = A[i]; A[i] = A[j]; A[j] = temp; // Перестановка
            temp = index[i]; index[i] = index[j]; index[j] = temp;
            i++; // Двигаемся дальше
            j--;
        }
    }
    quick_sort(A, from, j, index); // Сортируем левую часть
    quick_sort(A, i, to, index); // Сортируем правую часть
}


// ============================================================================================
void quick_sort(double* A, int from, int to, int* index)
{
    int i, j, index_temp;
    double temp, x;

    if (from >= to) return; // Условие окончания рекурсии
    i = from; // Рассматриваем элементы с A[from] до A[to]
    j = to;
    x = A[(from + to) / 2]; // Выбираем средний элемент
```

```
    while ( i <= j ) {
        while (A[i] < x) i++; // Ищем пару для перестановки
        while (A[j] > x) j--;
        if (i <= j) {
            temp = A[i]; A[i] = A[j]; A[j] = temp; // Перестановка
            index_temp = index[i]; index[i] = index[j]; index[j] = index_temp;
            i++; // Двигаемся дальше
            j--;
        }
    }
    quick_sort(A, from, j, index); // Сортируем левую часть
    quick_sort(A, i, to, index); // Сортируем правую часть
}


// ========================================================================================
void quick_sort(char A[][256], int from, int to, int* index)
{
    int i, j, index_temp;
    char temp[256], x[256];

    if (from >= to) return; // Условие окончания рекурсии
    i = from; // Рассматриваем элементы с A[from] до A[to]
    j = to;

    strcpy(x, A[(from + to) / 2]); // Выбираем средний элемент
    while ( i <= j ) {
        while (strcmp(A[i], x) < 0) i++; // Ищем пару для перестановки
        while (strcmp(A[j], x) > 0) j--;
        if (i <= j) {
            strcpy(temp, A[i]); strcpy(A[i], A[j]); strcpy(A[j], temp); // Перестановка
            index_temp = index[i]; index[i] = index[j]; index[j] = index_temp;
            i++; // Двигаемся дальше
            j--;
        }
    }
    quick_sort(A, from, j, index); // Сортируем левую часть
    quick_sort(A, i, to, index); // Сортируем правую часть
}
```