**ПРИЛОЖЕНИЕ А**

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

# КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

ОБРАБОТКА МАССИВОВ СТРУКТУРИРОВАННЫХ ДАННЫХ

**«**СТРАНЫ МИРА**»**

**ТЕКСТ ПРОГРАММЫ**

КР.АС59.200054-01 81 00

|  |  |
| --- | --- |
| 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>

int main()

{

1. int train\_count = 0; // Количество поездов на данный момент
2. Train\* train\_station = new Train[train\_count]; // Массив структур
3. int train\_buffer\_count = 0; // Количество действий в буфере
4. TrainBuffer\* trains\_buffer = new TrainBuffer[10]; // Буфер для хранения последних действий
5. // Массив указателей на функции
6. void(\*menu\_func\_arr[])(Train\*&, TrainBuffer\*, int&, int&) = {print\_train, add\_train, change\_train, delete\_train, sort\_train, select\_train, undo\_action};
7. int is\_file\_not\_open = 1; // Флаг состояния открытия файла (изначально не открыт)
8. char file\_name[256]; // Строка с именем файла
9. clear();
10. while (is\_file\_not\_open) { // Цикл открытия файла
    1. std::cout << "Введите имя файла: ";
    2. std::cin.getline(file\_name, 256); // Ввод имени файла
    3. is\_file\_not\_open = get\_records\_from\_file(file\_name, train\_station, train\_count); // Получение записей из файла и флага состояния
    4. if (is\_file\_not\_open) { // Если файл не открыт
       1. bool is\_false = true;
       2. while (is\_false) {
          1. int create;
          2. std::cout << "Создать новый файл? (0 - не cоздавать, 1 - создать): ";
          3. std::cin >> create;
          4. if (create < 0 || create > 1) {
             1. std::cout << "Неправильный ввод!!!";
             2. is\_false = true;
          5. }
          6. else {
             1. if (create == 0) {

is\_false = false;

* + - * 1. }
        2. 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);

* + - * 1. }
      1. }
    1. }
  1. }

1. }
2. create\_index\_file(train\_station, train\_count); // Создание индексных файлов
3. bool is\_working = true; // Флаг работы программы
4. while (is\_working) { // Цикл работы программы
   1. print\_menu(); // Вывод меню действий на экран
   2. int action; // Переменная действия
   3. while (true) {
      1. std::cout << "Выберите действие: ";
      2. std::cin >> action;
      3. if (std::cin.fail()) { // Если ввод был неудачным
         1. std::cin.clear();
         2. std::cin.ignore(32767, '\n');
         3. std::cout << "Неправильный ввод!!!" << std::endl;
      4. }
      5. else {
         1. std::cin.ignore(32767, '\n');
         2. break;
      6. }
   4. }
   5. clear();
   6. if (action >= 0 and action <= 7) { // Проверка действия
      1. if (action == 0) { // Отмена
         1. is\_working = false;
      2. }
      3. else {
         1. menu\_func\_arr[action - 1](train\_station, trains\_buffer, train\_count, train\_buffer\_count); // Вызов функции
      4. }
   7. }
   8. else {
      1. std::cout << "Такого действия нет" << std::endl;
   9. }
   10. put\_records\_in\_file(file\_name, train\_station, train\_count); // Запись данных в файл
5. }
6. delete[] train\_station; // Очистка пямяти
7. for (auto buffer: std::span(trains\_buffer, train\_buffer\_count)) { // Очистка буфера отмены
   1. delete[] buffer.trains;
8. }
9. delete[] trains\_buffer;
10. return 0;

}

**train.h**

#ifndef TRAIN\_H

#define TRAIN\_H

// Структура "Поезд"

#pragma pack(push, 1)

struct Train {

1. char number[5];
2. char end\_station[256];
3. char days[64];
4. char time\_departure[7];
5. char time\_way[7];
6. int stop\_count;

};

// Структура буфера для отмены действий

struct TrainBuffer {

1. int count;
2. 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)

{

1. std::cout << " Номер" << '|' << " Конечная станция" << '|' << " Дни следования" << '|' << " Отправление" << '|' << "Время в пути" << '|' << " Кол-во остановок" << '|' << "Время в пути в сутках";
2. std::cout << std::endl;
3. for (auto train: std::span(trains, count)) { // Цикл по всем записям
   1. std::cout << std::setw(7) << train.number << '|';
   2. std::cout << std::setw(20) << train.end\_station << '|';
   3. std::cout << std::setw(20) << train.days << '|';
   4. std::cout << std::setw(12) << train.time\_departure << '|';
   5. std::cout << std::setw(12) << train.time\_way << '|';
   6. std::cout << std::setw(17) << train.stop\_count << '|';
   7. // Расчет поля время в пути в сутках
   8. char hours\_str1[3] = " ", minutes\_str1[3] = " "; // Промежуточные переменные для часов и минут
   9. int end\_ch;
   10. double time\_in\_hours;
   11. for (int ch = 0; ch < strlen(train.time\_way); ch++) { // Запись часов записи
       1. if (train.time\_way[ch] != ':') { // Сравнение с двоеточием
          1. hours\_str1[ch] = train.time\_way[ch];
       2. }
       3. else { // Если равны, то записываем индекс
          1. end\_ch = ch + 1;
          2. break;
       4. }
   12. }
   13. minutes\_str1[0] = train.time\_way[end\_ch]; // Запись минут записи
   14. minutes\_str1[1] = train.time\_way[end\_ch + 1];
   15. time\_in\_hours = float(std::stoi(hours\_str1)) + float(std::stoi(minutes\_str1)) / 60.0; // Подсчёт времени записи
   16. std::cout << std::setw(21) << time\_in\_hours / 24.;
   17. std::cout << std::endl;
4. }

}

// Добавление нового поезда

// ======================================================================================

void add\_train(Train\* &trains, TrainBuffer\* trains\_buffer, int& count, int& buffer\_count)

{

1. print\_train(trains, trains\_buffer, count, buffer\_count);
2. bool is\_false = true;
3. save\_trains\_in\_buffer(trains, trains\_buffer, count, buffer\_count);
4. add\_memory\_train(trains, count);
5. int pos; // Позиция для добавления
6. while (true) {
   1. std::cout << "Введите позицию для добавления записи после указанной записи (-1 - в конец): ";
   2. std::cin >> pos;
   3. if (std::cin.fail()) {
      1. std::cin.clear();
      2. std::cin.ignore(32767, '\n');
      3. std::cout << "Неправильный ввод!!!" << std::endl;
   4. }
   5. else {
      1. std::cin.ignore(32767, '\n');
      2. if (pos < -1) {
         1. std::cout << "Позиция для добавления должна быть больше или равна -1!!!" << std::endl;
      3. }
      4. else if (pos == -1 || pos > count) { // Проверка на добавление в конец
         1. pos = count;
         2. break;
      5. }
      6. else {
         1. for (int i = count - 1; i >= pos; i--) { // Сдвиг элементов на один вперед
            1. trains[i + 1] = trains[i];
         2. }
         3. break;
      7. }
   6. }
7. }
8. while (is\_false) { // Проверка на правильность ввода
   1. std::cout << "Номер поезда: ";
   2. char input\_buffer[1024];
   3. std::cin.getline(input\_buffer, 1024);
   4. is\_false = false;
   5. if (strlen(input\_buffer) > 4) {
      1. std::cout << "Номер поезда должен состоять из менее, чем 5 символов!!!" << std::endl;
      2. is\_false = true;
      3. continue;
   6. }
   7. else {
      1. for (int i = 0; i < strlen(input\_buffer); i++) {
         1. if (int(input\_buffer[i]) < 48 || int(input\_buffer[i]) > 57) {
            1. is\_false = true;
            2. std::cout << "Номер должен состоять из цифр!!!" << std::endl;
            3. break;
         2. }
      2. }
      3. if (is\_false) {
         1. continue;
      4. }
      5. strcpy(trains[pos].number, input\_buffer);
   8. }
   9. for (int i = 0; auto train: std::span(trains, count)) { // Цикл по всем записям
      1. if (i == pos) continue;
      2. if (strcmp(trains[pos].number, train.number) == 0) { // Проверка на равенство номеров
         1. is\_false = true;
         2. std::cout << "Такой номер уже есть в базе данных!!!" << std::endl;
      3. }
      4. i++;
   10. }
9. }
10. std::cout << "Конечная станция: ";
11. std::cin.getline(trains[pos].end\_station, 256);
12. std::cout << "Дни следования: ";
13. std::cin.getline(trains[pos].days, 64);
14. is\_false = true;
15. while (is\_false) {
    1. std::cout << "Время отправления: ";
    2. std::cin.getline(trains[pos].time\_departure, 7);
    3. char hours\_str[3] = " ", minutes\_str[3] = " "; // Промежуточные переменные для часов и минут
    4. int end\_ch;
    5. for (int ch = 0; ch < strlen(trains[pos].time\_departure); ch++) { // Запись часов записи
       1. if (trains[pos].time\_departure[ch] != ':' && trains[pos].time\_departure[ch] != '.') { // Сравнение с двоеточием
          1. hours\_str[ch] = trains[pos].time\_departure[ch];
       2. }
       3. else { // Если равны, то записываем индекс
          1. if (trains[pos].time\_departure[ch] == '.') {
             1. trains[pos].time\_departure[ch] = ':';
          2. }
          3. end\_ch = ch + 1;
          4. break;
       4. }
    6. }
    7. minutes\_str[0] = trains[pos].time\_departure[end\_ch]; // Запись минут записи
    8. minutes\_str[1] = trains[pos].time\_departure[end\_ch + 1];
    9. int hours = std::stoi(hours\_str); // Перевод в целое число
    10. int mins = std::stoi(minutes\_str);
    11. if (hours < 0 || mins < 0 || mins >= 60) { // Проверка условий
        1. std::cout << "Неправильно задано время!!!" << std::endl;
        2. is\_false = true;
    12. }
    13. else {
        1. is\_false = false;
    14. }
16. }
17. is\_false = true;
18. while (is\_false) {
    1. std::cout << "Время в пути: ";
    2. std::cin.getline(trains[pos].time\_way, 7);
    3. char hours\_str[3] = " ", minutes\_str[3] = " "; // Промежуточные переменные для часов и минут
    4. int end\_ch;
    5. for (int ch = 0; ch < strlen(trains[pos].time\_way); ch++) { // Запись часов записи
       1. if (trains[pos].time\_way[ch] != ':' && trains[pos].time\_way[ch] != '.') { // Сравнение с двоеточием
          1. hours\_str[ch] = trains[pos].time\_way[ch];
       2. }
       3. else { // Если равны, то записываем индекс
          1. if (trains[pos].time\_way[ch] == '.') {
             1. trains[pos].time\_way[ch] = ':';
          2. }
          3. end\_ch = ch + 1;
          4. break;
       4. }
    6. }
    7. minutes\_str[0] = trains[pos].time\_way[end\_ch]; // Запись минут записи
    8. minutes\_str[1] = trains[pos].time\_way[end\_ch + 1];
    9. int hours = std::stoi(hours\_str); // Перевод в целое число
    10. int mins = std::stoi(minutes\_str);
    11. if (hours < 0 || mins < 0 || mins >= 60) { // Проверка условий
        1. std::cout << "Неправильно задано время!!!" << std::endl;
        2. is\_false = true;
    12. }
    13. else {
        1. is\_false = false;
    14. }
19. }
20. is\_false = true;
21. while (true) { // Проверка на правильность ввода
    1. std::cout << "Количество остановок: ";
    2. std::cin >> trains[pos].stop\_count;
    3. if (std::cin.fail()) {
       1. std::cin.clear();
       2. std::cin.ignore(32767, '\n');
       3. std::cout << "Неправильный ввод!!!" << std::endl;
    4. }
    5. else {
       1. std::cin.ignore(32767, '\n');
       2. if (trains[pos].stop\_count <= 0) {
          1. std::cout << "Количество остановок должно быть больше нуля!!!" << std::endl;
       3. }
       4. else {
          1. break;
       5. }
    6. }
22. }
23. count++; // Увеличение количества поездов на 1
24. create\_index\_file(trains, count);

}

// Функция изменения полей записи

// ======================================================================================

void change\_train(Train\* &trains, TrainBuffer\* trains\_buffer, int& count, int& buffer\_count)

{

1. char \_number[5]; // Номер записи для изменения
2. bool is\_stock = false; // Флаг наличия номера
3. int \_index = 0; // Индекс найденного элемента
4. save\_trains\_in\_buffer(trains, trains\_buffer, count, buffer\_count);
5. while (!is\_stock) { // Проверка на правильность ввода
   1. print\_train(trains, trains\_buffer, count, buffer\_count);
   2. std::cout << "Введите номер поезда для изменения: ";
   3. std::cin.getline(\_number, 5);
   4. for (int i = 0; auto train: std::span(trains, count)) { // Цикл по всем записям
      1. if (strcmp(\_number, train.number) == 0) { // Проверка на равенство номеров
         1. is\_stock = true;
         2. \_index = i;
         3. break;
      2. }
      3. i++;
   5. }
   6. clear();
   7. if (!is\_stock) {
      1. std::cout << "Такого номера нет!!!" << std::endl;
   8. }
6. }
7. bool is\_working = true; // Флаг работы программы
8. while (is\_working) {
   1. int action;
   2. std::cout << " Номер" << '|' << " Конечная станция" << '|' << " Дни следования" << '|' << " Отправление" << '|' << "Время в пути" << '|' << " Кол-во остановок" << '|' << "Время в пути в сутках";
   3. std::cout << std::endl;
   4. std::cout.width(7);
   5. std::cout << trains[\_index].number;
   6. std::cout << '|';
   7. std::cout.width(20);
   8. std::cout << trains[\_index].end\_station;
   9. std::cout << '|';
   10. std::cout.width(20);
   11. std::cout << trains[\_index].days;
   12. std::cout << '|';
   13. std::cout.width(12);
   14. std::cout << trains[\_index].time\_departure;
   15. std::cout << '|';
   16. std::cout.width(12);
   17. std::cout << trains[\_index].time\_way;
   18. std::cout << '|';
   19. std::cout.width(17);
   20. std::cout << trains[\_index].stop\_count;
   21. char hours\_str1[3] = " ", minutes\_str1[3] = " "; // Промежуточные переменные для часов и минут
   22. int end\_ch;
   23. double time\_in\_hours;
   24. for (int ch = 0; ch < strlen(trains[\_index].time\_way); ch++) { // Запись часов записи
       1. if (trains[\_index].time\_way[ch] != ':') { // Сравнение с двоеточием
          1. hours\_str1[ch] = trains[\_index].time\_way[ch];
       2. }
       3. else { // Если равны, то записываем индекс
          1. end\_ch = ch + 1;
          2. break;
       4. }
   25. }
   26. minutes\_str1[0] = trains[\_index].time\_way[end\_ch]; // Запись минут записи
   27. minutes\_str1[1] = trains[\_index].time\_way[end\_ch + 1];
   28. time\_in\_hours = float(std::stoi(hours\_str1)) + float(std::stoi(minutes\_str1)) / 60.0; // Подсчёт времени записи
   29. std::cout << '|';
   30. std::cout.width(21);
   31. std::cout << time\_in\_hours / 24.;
   32. std::cout << std::endl;
   33. print\_change\_menu();
   34. while (true) {
       1. std::cout << "Выберите действие: ";
       2. std::cin >> action;
       3. if (std::cin.fail()) {
          1. std::cin.clear();
          2. std::cin.ignore(32767, '\n');
          3. std::cout << "Неправильный ввод!!!" << std::endl;
       4. }
       5. else {
          1. std::cin.ignore(32767, '\n');
          2. break;
       6. }
   35. }
   36. clear();
   37. bool is\_false = true;
   38. switch (action) {
       1. case 1: // Изменение номера
          1. is\_false = true;
          2. while (is\_false) { // Проверка на правильность ввода
             1. std::cout << "Номер поезда:";
             2. char input\_buffer[1024];
             3. std::cin.getline(input\_buffer, 1024);
             4. is\_false = false;
             5. if (strlen(input\_buffer) > 4) {

std::cout << "Номер поезда должен состоять из менее, чем 5 символов!!!" << std::endl;

is\_false = true;

continue;

* + - * 1. }
        2. 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);

* + - * 1. }
        2. 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++;

* + - * 1. }
      1. }
      2. break;
    1. case 2: // Изменение конечной станции
       1. std::cout << "введите новое название конечной станции: ";
       2. std::cin.getline(trains[\_index].end\_station, 256);
       3. break;
    2. case 3: // Изменение дней следования
       1. std::cout << "Введите новые дни следования: ";
       2. std::cin.getline(trains[\_index].days, 64);
       3. break;
    3. case 4: // Изменение времени отправления
       1. is\_false = true;
       2. while (is\_false) {
          1. std::cout << "Введите новое время отправления: ";
          2. std::cin.getline(trains[\_index].time\_departure, 7);
          3. char hours\_str[3] = " ", minutes\_str[3] = " "; // Промежуточные переменные для часов и минут
          4. int end\_ch;
          5. 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;

}

* + - * 1. }
        2. minutes\_str[0] = trains[\_index].time\_departure[end\_ch]; // Запись минут записи
        3. minutes\_str[1] = trains[\_index].time\_departure[end\_ch + 1];
        4. int hours = std::stoi(hours\_str); // Перевод в целое число
        5. int mins = std::stoi(minutes\_str);
        6. if (hours < 0 || mins < 0 || mins >= 60) { // Проверка условий

std::cout << "Неправильно задано время!!!" << std::endl;

is\_false = true;

* + - * 1. }
        2. else {

is\_false = false;

* + - * 1. }
      1. }
      2. break;
    1. case 5: // Изменение времени в пути
       1. is\_false = true;
       2. while (is\_false) {
          1. std::cout << "Введите новое время в пути: ";
          2. std::cin.getline(trains[\_index].time\_way, 7);
          3. char hours\_str[3] = " ", minutes\_str[3] = " "; // Промежуточные переменные для часов и минут
          4. int end\_ch;
          5. 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;

}

* + - * 1. }
        2. minutes\_str[0] = trains[\_index].time\_way[end\_ch]; // Запись минут записи
        3. minutes\_str[1] = trains[\_index].time\_way[end\_ch + 1];
        4. int hours = std::stoi(hours\_str); // Перевод в целое число
        5. int mins = std::stoi(minutes\_str);
        6. if (hours < 0 || mins < 0 || mins >= 60) { // Проверка условий

std::cout << "Неправильно задано время!!!" << std::endl;

is\_false = true;

* + - * 1. }
        2. else {

is\_false = false;

* + - * 1. }
      1. }
      2. break;
    1. case 6: // Изменение количества остановок
       1. while (true) {
          1. std::cout << "Введите новое количество остановок: ";
          2. std::cin >> trains[\_index].stop\_count;
          3. if (std::cin.fail()) {

std::cin.clear();

std::cin.ignore(32767, '\n');

std::cout << "Неправильный ввод!!!" << std::endl;

* + - * 1. }
        2. 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;

}

* + - * 1. }
      1. }
      2. break;
    1. case 0: // Завершить изменение
       1. is\_working = false;
       2. break;
    2. default:
       1. std::cout << "Такого действия нет" << std::endl;
       2. break;
  1. }

1. }
2. create\_index\_file(trains, count);

}

// Удаление поезда по номеру в таблице

// ======================================================================================

void delete\_train(Train\* &trains, TrainBuffer\* trains\_buffer, int& count, int& buffer\_count)

{

1. print\_train(trains, trains\_buffer, count, buffer\_count);
2. print\_delete\_menu();
3. int type; // Переменная типа удаления
4. while (true) {
   1. std::cout << "Выберите тип удаления: ";
   2. std::cin >> type;
   3. if (std::cin.fail()) {
      1. std::cin.clear();
      2. std::cin.ignore(32767, '\n');
      3. std::cout << "Неправильный ввод!!!" << std::endl;
   4. }
   5. else {
      1. std::cin.ignore(32767, '\n');
      2. if (type < 0 || type > 5) {
         1. std::cout << "Такого типа нет!!!" << std::endl;
      3. }
      4. else {
         1. break;
      5. }
   6. }
5. }
6. switch (type) { // Выбор по типу удаления
   1. case 1: // Удаление по номеру
      1. char number[5];
      2. std::cout << "Введите номер: ";
      3. std::cin.getline(number, 5);
      4. save\_trains\_in\_buffer(trains, trains\_buffer, count, buffer\_count);
      5. delete\_train\_by\_number(trains, count, number);
      6. break;
   2. case 2: // Удаление по конечной станции
      1. char end\_station[256];
      2. std::cout << "Введите название станции: ";
      3. std::cin.getline(end\_station, 256);
      4. save\_trains\_in\_buffer(trains, trains\_buffer, count, buffer\_count);
      5. delete\_train\_by\_end\_station(trains, count, end\_station);
      6. break;
   3. case 3: // Удаление по времени отправления
      1. char departure\_time[7];
      2. std::cout << "Введите время: ";
      3. std::cin.getline(departure\_time, 7);
      4. save\_trains\_in\_buffer(trains, trains\_buffer, count, buffer\_count);
      5. delete\_train\_by\_departure\_time(trains, count, departure\_time);
      6. break;
   4. case 4: // Удаление по времени в пути
      1. char way\_time[7];
      2. std::cout << "Введите время: ";
      3. std::cin.getline(way\_time, 7);
      4. save\_trains\_in\_buffer(trains, trains\_buffer, count, buffer\_count);
      5. delete\_train\_by\_way\_time(trains, count, way\_time);
      6. break;
   5. case 5: // Удаление по количеству остановок
      1. int stop\_count;
      2. while (true) { // Проверка на правильность ввода
         1. std::cout << "Введите количество остановок: ";
         2. std::cin >> stop\_count;
         3. if (std::cin.fail()) {
            1. std::cin.clear();
            2. std::cin.ignore(32767, '\n');
            3. std::cout << "Неправильный ввод!!!" << std::endl;
         4. }
         5. else {
            1. std::cin.ignore(32767, '\n');
            2. if (stop\_count <= 0) {

std::cout << "Количество остановок должно быть больше нуля!!!" << std::endl;

* + - * 1. }
        2. else {

break;

* + - * 1. }
      1. }
    1. }
    2. save\_trains\_in\_buffer(trains, trains\_buffer, count, buffer\_count);
    3. delete\_train\_by\_stop\_count(trains, count, stop\_count);
    4. break;
  1. case 0: // Отмена
     1. break;

1. }
2. create\_index\_file(trains, count);

}

// Сортировка поездов

// ======================================================================================

void sort\_train(Train\*& trains, TrainBuffer\* trains\_buffer, int& count, int& buffer\_count)

{

1. int type; // Переменная типа сортировки
2. bool reverse; // Флаг направления сортировки (прямая / обратная)
3. bool in\_file; // Флаг записи в файл
4. print\_sort\_menu(); // Вывод меню типов сортировки
5. while (true) {
   1. std::cout << "Выберите тип сортировки: ";
   2. std::cin >> type;
   3. if (std::cin.fail()) {
      1. std::cin.clear();
      2. std::cin.ignore(32767, '\n');
      3. std::cout << "Неправильный ввод!!!" << std::endl;
   4. }
   5. else {
      1. std::cin.ignore(32767, '\n');
      2. if (type < 0 || type > 5) {
         1. std::cout << "Такого типа нет!!!" << std::endl;
      3. }
      4. else {
         1. break;
      5. }
   6. }
6. }
7. print\_reverse\_menu(); // Вывод меню флагов сортировки
8. while (true) {
   1. std::cout << "Выберите порядок сортировки: ";
   2. std::cin >> reverse;
   3. if (std::cin.fail()) {
      1. std::cin.clear();
      2. std::cin.ignore(32767, '\n');
      3. std::cout << "Неправильный ввод!!!" << std::endl;
   4. }
   5. else {
      1. std::cin.ignore(32767, '\n');
      2. break;
   6. }
9. }
10. print\_file\_menu(); // Вывод меню флагов записи в файл
11. while (true) {
    1. std::cout << "Выберите тип записи в файл: ";
    2. std::cin >> in\_file;
    3. if (std::cin.fail()) {
       1. std::cin.clear();
       2. std::cin.ignore(32767, '\n');
       3. std::cout << "Неправильный ввод!!!" << std::endl;
    4. }
    5. else {
       1. std::cin.ignore(32767, '\n');
       2. break;
    6. }
12. }
13. Train\* sort\_trains = new Train[count]; // Создание временного массива поездов
14. for (int i = 0; auto train: std::span(trains, count)) { // Переписываем все поезда из оригинального массива
    1. sort\_trains[i] = train;
    2. i++;
15. }
16. std::ifstream file;
17. int index[count];
18. bool is\_end = false;
19. switch (type) { // Выбор по типу сортировки
    1. case 1: // Сортировка по номеру
       1. file.open("index\_file/number\_sort.txt", std::ios::binary);
       2. break;
    2. case 2: // Сортировка по названию станции
       1. file.open("index\_file/end\_station\_sort.txt", std::ios::binary);
       2. break;
    3. case 3: // Сортировка по времени отправления
       1. file.open("index\_file/departure\_time\_sort.txt", std::ios::binary);
       2. break;
    4. case 4: // Сортировка по времени в пути
       1. file.open("index\_file/way\_time\_sort.txt", std::ios::binary);
       2. break;
    5. case 5: // Сортировка по количеству остановок
       1. file.open("index\_file/stop\_count\_sort.txt", std::ios::binary);
       2. break;
    6. case 0: // Отмена
       1. is\_end = true;
       2. break;
20. }
21. if (!is\_end) {
    1. // Чтение индексного файла
    2. file.read((char\*)&count, sizeof(int));
    3. for (auto & i: std::span(index, count)) {
       1. file.read((char\*)(&i), sizeof(int));
    4. }
    5. // Сортировка в срртветствии с индексами
    6. int index\_index = (reverse) ? Count - 1 : 0;
    7. for (int i = 0; i < count; i++) {
       1. sort\_trains[i] = trains[index[index\_index]];
       2. if (reverse) {
          1. index\_index--;
       3. }
       4. else {
          1. index\_index++;
       5. }
    8. }
    9. print\_train(sort\_trains, trains\_buffer, count, buffer\_count); // Вывод поездов на экран
    10. if (in\_file) {
        1. save\_trains\_in\_buffer(trains, trains\_buffer, count, buffer\_count);
        2. for (int i = 0; auto train: std::span(sort\_trains, count)) {
           1. trains[i] = train;
           2. i++;
        3. }
        4. create\_index\_file(trains, count);
    11. }
22. }
23. delete[] sort\_trains; // Очищение выделенной памяти

}

// Функция выборки записей по полям

// ======================================================================================

void select\_train(Train\*& trains, TrainBuffer\* trains\_buffer, int& count, int& buffer\_count)

{

1. int type; // Тип выборки/поиска
2. print\_train(trains, trains\_buffer, count, buffer\_count);
3. print\_selection\_menu();
4. while (true) {
   1. std::cout << "Выберите тип выборки: ";
   2. std::cin >> type;
   3. if (std::cin.fail()) {
      1. std::cin.clear();
      2. std::cin.ignore(32767, '\n');
      3. std::cout << "Неправильный ввод!!!" << std::endl;
   4. }
   5. else {
      1. std::cin.ignore(32767, '\n');
      2. if (type < 0 || type > 5) {
         1. std::cout << "Такого типа нет!!!" << std::endl;
      3. }
      4. else {
         1. break;
      5. }
   6. }
5. }
6. switch (type) {
   1. case 1: // Выборка по номеру
      1. int down\_number, up\_number; // Диапазон номеров
      2. std::cout << "Введите диапазон значений" << std::endl;
      3. while (true) { // Проверка на правильность ввода
         1. std::cout << "Нижняя граница: ";
         2. std::cin >> down\_number;
         3. if (std::cin.fail()) {
            1. std::cin.clear();
            2. std::cin.ignore(32767, '\n');
            3. std::cout << "Неправильный ввод!!!" << std::endl;
         4. }
         5. else {
            1. std::cin.ignore(32767, '\n');
            2. if (down\_number > 9999 || down\_number <= 0) {

std::cout << "Номер поезда должен состоять из менее, чем 5 символов, и быть больше нуля!!!" << std::endl;

* + - * 1. }
        2. else {

break;

* + - * 1. }
      1. }
    1. }
    2. while (true) { // Проверка на правильность ввода
       1. std::cout << "Верхняя граница: ";
       2. std::cin >> up\_number;
       3. if (std::cin.fail()) {
          1. std::cin.clear();
          2. std::cin.ignore(32767, '\n');
          3. std::cout << "Неправильный ввод!!!" << std::endl;
       4. }
       5. else {
          1. std::cin.ignore(32767, '\n');
          2. if (up\_number > 9999 || up\_number <= 0) {
          3. std::cout << "Номер поезда должен состоять из менее, чем 5 символов, и быть больше нуля!!!" << std::endl;
       6. }
       7. else {
          1. break;
       8. }
    3. }
  1. }
  2. select\_by\_number(trains, trains\_buffer, count, buffer\_count, down\_number, up\_number);
  3. break;

1. case 2: // Выборка по названию конечной станции
   1. char end\_station[256];
   2. std::cout << "Введите название конечной станции: ";
   3. std::cin.getline(end\_station, 256);
   4. select\_by\_end\_station(trains, trains\_buffer, count, buffer\_count, end\_station);
   5. break;
2. case 3: // Выборка по времени отправления
   1. char down\_departure\_time[7], up\_departure\_time[7]; // Диапазон веремени отправления
   2. std::cout << "Введите диапазон значений" << std::endl;
   3. std::cout << "Нижняя граница: ";
   4. std::cin.getline(down\_departure\_time, 7);
   5. std::cout << "Верхняя граница: ";
   6. std::cin.getline(up\_departure\_time, 7);
   7. select\_by\_departure\_time(trains, trains\_buffer, count, buffer\_count, down\_departure\_time, up\_departure\_time);
   8. break;
3. case 4: // Выборка по времени в пути
   1. char down\_way\_time[7], up\_way\_time[7]; // Диапазон времени в пути
   2. std::cout << "Введите диапазон значений" << std::endl;
   3. std::cout << "Нижняя граница: ";
   4. std::cin.getline(down\_way\_time, 7);
   5. std::cout << "Верхняя граница: ";
   6. std::cin.getline(up\_way\_time, 7);
   7. select\_by\_way\_time(trains, trains\_buffer, count, buffer\_count, down\_way\_time, up\_way\_time);
   8. break;
4. case 5: // Выборка по количеству остановок
   1. int down\_stop\_number, up\_stop\_number; // Диапазон количества остановок
   2. std::cout << "Введите диапазон значений" << std::endl;
   3. while (true) { // Проверка на правильность ввода
      1. std::cout << "Нижняя граница: ";
      2. std::cin >> down\_stop\_number;
      3. if (std::cin.fail()) {
         1. std::cin.clear();
         2. std::cin.ignore(32767, '\n');
         3. std::cout << "Неправильный ввод!!!" << std::endl;
      4. }
      5. else {
         1. std::cin.ignore(32767, '\n');
         2. if (down\_stop\_number <= 0) {
            1. std::cout << "Количество остановок должно быть больше нуля!!!" << std::endl;
         3. }
         4. else {
            1. break;
         5. }
      6. }
   4. }
   5. while (true) { // Проверка на правильность ввода
      1. std::cout << "Верхняя граница: ";
      2. std::cin >> up\_stop\_number;
      3. if (std::cin.fail()) {
         1. std::cin.clear();
         2. std::cin.ignore(32767, '\n');
         3. std::cout << "Неправильный ввод!!!" << std::endl;
      4. }
      5. else {
         1. std::cin.ignore(32767, '\n');
         2. if (up\_stop\_number <= 0) {
            1. std::cout << "Количество остановок должно быть больше нуля!!!" << std::endl;
         3. }
         4. else {
            1. break;
         5. }
      6. }
   6. }
   7. select\_by\_stop\_count(trains, trains\_buffer, count, buffer\_count, down\_stop\_number, up\_stop\_number);
   8. break;
5. case 0: // Отмена
   1. break;
6. }

}

// Функция отмены последнего действия

// ======================================================================================

void undo\_action(Train\* &trains, TrainBuffer\* trains\_buffer, int& count, int& buffer\_count)

{

1. if (buffer\_count != 0) { // Проверка на наличие состояний для отмены
   1. count = trains\_buffer[buffer\_count - 1].count; // Получение количества поездов
   2. delete[] trains; // Удаление указателя
   3. trains = new Train[count]; // Создание нового с новым количеством
   4. for (int i = 0; auto train: std::span(trains\_buffer[buffer\_count - 1].trains, count)) {
      1. trains[i] = train; // Перепись данных из буфера в массив
      2. i++;
   5. }
   6. delete[] trains\_buffer[buffer\_count - 1].trains; // Удаление буфера
   7. buffer\_count--; // Уменьшение количества сохраненных состояний
   8. create\_index\_file(trains, count);
2. }

}

**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()

{

1. std::cout << std::endl;
2. std::cout << "=====================MENU=====================" << std::endl;
3. std::cout << std::endl;
4. std::cout
5. << "1" << '\t' << "вывод всех поездов" << std::endl
6. << "2" << '\t' << "добавление нового поезда" << std::endl
7. << "3" << '\t' << "изменение поезда" << std::endl
8. << "4" << '\t' << "удаление поезда" << std::endl
9. << "5" << '\t' << "сортировка поездов" << std::endl
10. << "6" << '\t' << "выборка записей" << std::endl
11. << "7" << '\t' << "отмена последнего действия" << std::endl
12. << "0" << '\t' << "выход из программы" << std::endl;
13. std::cout << std::endl;

}

// Вывод меню типов сортировки

// ======================================================================================

void print\_sort\_menu()

{

1. std::cout << std::endl;
2. std::cout << "==================SORT MENU===================" << std::endl;
3. std::cout << std::endl;
4. std::cout
5. << "1" << '\t' << "сортировка по номеру" << std::endl
6. << "2" << '\t' << "сортировка по конечной станции" << std::endl
7. << "3" << '\t' << "сортировка по времени отправления" << std::endl
8. << "4" << '\t' << "сортировка по времени в пути" << std::endl
9. << "5" << '\t' << "сортировка по количеству остановок" << std::endl
10. << "0" << '\t' << "отмена" << std::endl;
11. std::cout << std::endl;

}

// Вывод меню флагов сортировки

// ======================================================================================

void print\_reverse\_menu()

{

1. std::cout << std::endl;
2. std::cout << "===================REVERSE====================" << std::endl;
3. std::cout << std::endl;
4. std::cout
5. << "0" << '\t' << "прямая сортировка (от меньшего к большему)" << std::endl
6. << "1" << '\t' << "обратная сортировка (от большего к меньшему)" << std::endl;
7. std::cout << std::endl;

}

// Вывод меню флагов записи в файл

// ======================================================================================

void print\_file\_menu()

{

1. std::cout << std::endl;
2. std::cout << "=====================FILE=====================" << std::endl;
3. std::cout << std::endl;
4. std::cout
5. << "0" << '\t' << "без записи в файл" << std::endl
6. << "1" << '\t' << "с записью в файл" << std::endl;
7. std::cout << std::endl;

}

// Вывод меню типов удаления

// ======================================================================================

void print\_delete\_menu()

{

1. std::cout << std::endl;
2. std::cout << "====================DELETE====================" << std::endl;
3. std::cout << std::endl;
4. std::cout
5. << "1" << '\t' << "удаление записи по номеру" << std::endl
6. << "2" << '\t' << "удаление записей по названию станции" << std::endl
7. << "3" << '\t' << "удаление записей по времени отправления" << std::endl
8. << "4" << '\t' << "удаление записей по времени в пути" << std::endl
9. << "5" << '\t' << "удаление записей по количеству остановок" << std::endl
10. << "0" << '\t' << "отмена" << std::endl;
11. std::cout << std::endl;

}

// Вывод меню типов выборки

// ======================================================================================

void print\_selection\_menu()

{

1. std::cout << std::endl;
2. std::cout << "====================SELECT====================" << std::endl;
3. std::cout << std::endl;
4. std::cout
5. << "1" << '\t' << "выборка записей по номеру" << std::endl
6. << "2" << '\t' << "выборка записей по названию станции" << std::endl
7. << "3" << '\t' << "выборка записей по времени отправления" << std::endl
8. << "4" << '\t' << "выборка записей по времени в пути" << std::endl
9. << "5" << '\t' << "выборка записей по количеству остановок" << std::endl
10. << "0" << '\t' << "отмена" << std::endl;
11. std::cout << std::endl;

}

// Вывод меню изменения записи

// ======================================================================================

void print\_change\_menu()

{

1. std::cout << std::endl;
2. std::cout << "====================CHANGE====================" << std::endl;
3. std::cout << std::endl;
4. std::cout
5. << "1" << '\t' << "изменить номер" << std::endl
6. << "2" << '\t' << "изменить название станции" << std::endl
7. << "3" << '\t' << "изменить дни следования" << std::endl
8. << "4" << '\t' << "изменить время отправления" << std::endl
9. << "5" << '\t' << "изменить время в пути" << std::endl
10. << "6" << '\t' << "изменить количество остановок" << std::endl
11. << "0" << '\t' << "завершить изменение" << std::endl;
12. 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)

{

1. std::ifstream file(file\_name, std::ios::binary); // Создание потока чтения из файла
2. if (!file) { // Проверка на ошибки при открытии файла
   1. std::cout << "Файл не открыт!!!" << std::endl;
   2. return 1;
3. }
4. int train\_count\_in\_file;
5. file.read((char\*)(&train\_count\_in\_file), sizeof(train\_count\_in\_file));
6. for (int i = 0; i < train\_count\_in\_file; i++) { // Если файл открыт, то считываем данные
   1. add\_memory\_train(trains, count);
   2. file.read((char\*)(&trains[count]), sizeof(Train));
   3. count++;
7. }
8. return 0;

}

// Запись данных в файл

// ======================================================================================

int put\_records\_in\_file(char file\_name[], Train\* trains, int count)

{

1. std::ofstream file(file\_name, std::ios::binary); // Создание потока записи в файл
2. std::ofstream text\_file("output\_file.txt");
3. if (!file || !text\_file) {
   1. std::cout << "Файл не открыт!!!" << std::endl;
   2. return 1;
4. }
5. file.write((char\*)(&count), sizeof(count)); // Запись количества записей в файл
6. text\_file << " Номер" << '|' << " Конечная станция" << '|' << " Дни следования" << '|'
7. << " Отправление" << '|' << "Время в пути" << '|' << " Кол-во остановок" << '|' << "Время в пути в сутках";
8. text\_file << std::endl; // Запись шапки в текстовый файл
9. for (auto & train: std::span(trains, count)) { // Цикл по количеству записей
   1. file.write((char\*)(&train), sizeof(Train)); // Запись структуры в файл
   2. // Запись полей структуры в текстовый файл
   3. text\_file << std::setw(7) << train.number << '|';
   4. text\_file << std::setw(20) << train.end\_station << '|';
   5. text\_file << std::setw(20) << train.days << '|';
   6. text\_file << std::setw(12) << train.time\_departure << '|';
   7. text\_file << std::setw(12) << train.time\_way << '|';
   8. text\_file << std::setw(17) << train.stop\_count << '|';
   9. // Расчет поля время в пути в сутках
   10. char hours\_str1[3] = " ", minutes\_str1[3] = " "; // Промежуточные переменные для часов и минут
   11. int end\_ch;
   12. double time\_in\_hours;
   13. for (int ch = 0; ch < strlen(train.time\_way); ch++) { // Запись часов записи
       1. if (train.time\_way[ch] != ':') { // Сравнение с двоеточием
          1. hours\_str1[ch] = train.time\_way[ch];
       2. }
       3. else { // Если равны, то записываем индекс
          1. end\_ch = ch + 1;
          2. break;
       4. }
   14. }
   15. minutes\_str1[0] = train.time\_way[end\_ch]; // Запись минут записи
   16. minutes\_str1[1] = train.time\_way[end\_ch + 1];
   17. time\_in\_hours = float(std::stoi(hours\_str1)) + float(std::stoi(minutes\_str1)) / 60.0; // Подсчёт времени записи
   18. text\_file << std::setw(21) << time\_in\_hours / 24.;
   19. text\_file << std::endl;
10. }
11. return 0;

}

// Функция создания индексных файлов

// ======================================================================================

void create\_index\_file(Train\* trains, int count)

{

1. Train\* trains\_buffer = new Train[count];
2. int\* index = new int[count];
3. for (int i = 0; i < count; i++) {
   1. trains\_buffer[i] = trains[i];
4. }
5. number\_sort(trains\_buffer, index, count);
6. std::ofstream file("index\_file/number\_sort.txt", std::ios::binary);
7. file.write((char\*)(&count), sizeof(int));
8. for (int i = 0; i < count; i++) {
   1. file.write((char\*)(&index[i]), sizeof(int));
9. }
10. file.close();
11. for (int i = 0; i < count; i++) {
    1. trains\_buffer[i] = trains[i];
12. }
13. end\_station\_sort(trains\_buffer, index, count);
14. file.open("index\_file/end\_station\_sort.txt", std::ios::binary);
15. file.write((char\*)(&count), sizeof(int));
16. for (int i = 0; i < count; i++) {
    1. file.write((char\*)(&index[i]), sizeof(int));
17. }
18. file.close();
19. for (int i = 0; i < count; i++) {
    1. trains\_buffer[i] = trains[i];
20. }
21. departure\_time\_sort(trains\_buffer, index, count);
22. file.open("index\_file/departure\_time\_sort.txt", std::ios::binary);
23. file.write((char\*)(&count), sizeof(int));
24. for (int i = 0; i < count; i++) {
    1. file.write((char\*)(&index[i]), sizeof(int));
25. }
26. file.close();
27. for (int i = 0; i < count; i++) {
    1. trains\_buffer[i] = trains[i];
28. }
29. way\_time\_sort(trains\_buffer, index, count);
30. file.open("index\_file/way\_time\_sort.txt", std::ios::binary);
31. file.write((char\*)(&count), sizeof(int));
32. for (int i = 0; i < count; i++) {
    1. file.write((char\*)(&index[i]), sizeof(int));
33. }
34. file.close();
35. for (int i = 0; i < count; i++) {
    1. trains\_buffer[i] = trains[i];
36. }
37. stop\_count\_sort(trains\_buffer, index, count);
38. file.open("index\_file/stop\_count\_sort.txt", std::ios::binary);
39. file.write((char\*)(&count), sizeof(int));
40. for (int i = 0; i < count; i++) {
    1. file.write((char\*)(&index[i]), sizeof(int));
41. }
42. 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()

{

1. 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)

{

1. Train\* buffer\_trains = new Train[count]; // Создание временного буфера
2. for (int i = 0; auto train: std::span(trains, count)) { // Перепись данных в буфер
   1. buffer\_trains[i] = train;
   2. i++;
3. }
4. delete[] trains; // Удаление указателя
5. trains = new Train[count + 1]; // Создание нового указателя с большим количеством элементов
6. for (int i = 0; auto train: std::span(buffer\_trains, count)) { // Перепись данных из буфера в массив
   1. trains[i] = train;
   2. i++;
7. }
8. delete[] buffer\_trains; // Удаление буфера

}

// Функция очищения ячейки памяти

// ======================================================================================

void sub\_memory\_train(Train\* &trains, int& count)

{

1. Train\* buffer\_trains = new Train[count - 1]; // Создание временного буфера
2. for (int i = 0; i < count - 1; i++) { // Перепись данных в буфер
   1. buffer\_trains[i] = trains[i];
3. }
4. delete[] trains; // Удаление указателя
5. trains = new Train[count - 1]; // Создание нового указателя с меньшим количеством элементов
6. for (int i = 0; i < count - 1; i++) { // Перепись данных из буфера в массив
   1. trains[i] = buffer\_trains[i];
7. }
8. 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)

{

1. if (buffer\_count == 10) {
   1. delete[] trains\_buffer[0].trains; // Удаление первого состояния
   2. for (int i = 0; i < buffer\_count - 1; i++) { // Сдвиг всех элементов назад на один
      1. trains\_buffer[i] = trains\_buffer[i + 1];
   3. }
   4. buffer\_count = 9; // Изменение количества
2. }
3. trains\_buffer[buffer\_count].count = count; // Запись количества поездов
4. trains\_buffer[buffer\_count].trains = new Train[count]; // Запись поездов
5. for (int i = 0; auto train: std::span(trains, count)) { // Перепись поездов из массива в буфер для отмены
   1. trains\_buffer[buffer\_count].trains[i] = train;
   2. i++;
6. }
7. 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])

{

1. int i = 0;
2. while (i < count) {
   1. if (strcmp(trains[i].number, number) == 0) { // Сравнение номеров
      1. if (i != count - 1) {
         1. for (int j = i; j < count - 1; j++) { // Сдвиг записей на один назад
            1. trains[j] = trains[j + 1];
         2. }
         3. sub\_memory\_train(trains, count);
         4. count--;
         5. break;
      2. }
      3. else {
         1. sub\_memory\_train(trains, count);
         2. count--;
         3. break;
      4. }
   2. }
   3. i++;
3. }
4. 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])

{

1. int i = 0;
2. while (i < count) {
   1. bool is\_equal = true;
   2. if (strlen(trains[i].end\_station) < strlen(end\_station)) { // Проверка на длину
      1. is\_equal = false;
   3. }
   4. else {
      1. for (int j = 0; j < strlen(end\_station); j++) { // Сравнение названия станции
         1. if (trains[i].end\_station[j] != end\_station[j]) {
            1. is\_equal = false;
            2. break;
         2. }
      2. }
   5. }
   6. if (is\_equal) {
      1. for (int j = i; j < count - 1; j++) { // Сдвиг на один назад
         1. trains[j] = trains[j + 1];
      2. }
      3. i--;
      4. sub\_memory\_train(trains, count);
      5. count--;
   7. }
   8. i++;
3. }
4. create\_index\_file(trains, count);

}

// Удаление поездов по времени отправления

// ======================================================================================

void delete\_train\_by\_departure\_time(Train\* &trains, int& count, char departure\_time[7])

{

1. int i = 0;
2. while (i < count) {
   1. bool is\_equal = true;
   2. if (strlen(trains[i].time\_departure) < strlen(departure\_time)) { // Проверка на длину
      1. is\_equal = false;
   3. }
   4. else {
      1. for (int j = 0; j < strlen(departure\_time); j++) { // Сравнение времени отправления
         1. if (trains[i].time\_departure[j] != departure\_time[j]) {
            1. is\_equal = false;
            2. break;
         2. }
      2. }
   5. }
   6. if (is\_equal) {
      1. for (int j = i; j < count - 1; j++) { // Сдвиг на один назад
         1. trains[j] = trains[j + 1];
      2. }
      3. i--;
      4. sub\_memory\_train(trains, count);
      5. count--;
   7. }
   8. i++;
3. }
4. 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])

{

1. int i = 0;
2. while (i < count) {
   1. bool is\_equal = true;
   2. if (strlen(trains[i].time\_way) < strlen(way\_time)) { // Проверка на длину
      1. is\_equal = false;
   3. }
   4. else {
      1. for (int j = 0; j < strlen(way\_time); j++) { // Сравнение времени в пути
         1. if (trains[i].time\_way[j] != way\_time[j]) {
            1. is\_equal = false;
            2. break;
         2. }
      2. }
   5. }
   6. if (is\_equal) {
      1. for (int j = i; j < count - 1; j++) { // Сдвиг на один назад
         1. trains[j] = trains[j + 1];
      2. }
      3. i--;
      4. sub\_memory\_train(trains, count);
      5. count--;
   7. }
   8. i++;
3. }
4. create\_index\_file(trains, count);

}

// Удаление поездов по количеству остановок

// ======================================================================================

void delete\_train\_by\_stop\_count(Train\* &trains, int& count, int stop\_count)

{

1. int i = 0;
2. while (i < count) {
   1. if (trains[i].stop\_count == stop\_count) { // Сравнение количества остановок
      1. for (int j = i; j < count - 1; j++) { // Сдвиг на один назад
         1. trains[j] = trains[j + 1];
      2. }
      3. i--;
      4. sub\_memory\_train(trains, count);
      5. count--;
   2. }
   3. i++;
3. }
4. 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)

{

1. Train\* trains\_buffer = new Train[count]; // Буфер поездов для выборки
2. int selection\_count = 0; // Количество найденных поездов
3. for (int i = 0; auto train: std::span(trains, count)) { // Цикл по записям
   1. int \_number = std::stoi(train.number);
   2. if (\_number >= down\_number && \_number <= up\_number) { // Проверка в диапазоне
      1. trains\_buffer[selection\_count] = train;
      2. selection\_count++;
   3. }
   4. i++;
4. }
5. print\_train(trains\_buffer, trains\_buffer1, selection\_count, buffer\_count);
6. std::cout << "Количество записей найдено: " << selection\_count << std::endl;
7. 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])

{

1. Train\* trains\_buffer = new Train[count]; // Буфер поездов для выборки
2. int selection\_count = 0; // Количество найденных поездов
3. int i = 0;
4. while (i < count) { // Цикл по записям
   1. bool is\_equal = true;
   2. if (strlen(trains[i].end\_station) < strlen(end\_station)) { // Проверка на длину строки
      1. is\_equal = false;
   3. }
   4. else {
      1. for (int j = 0; j < strlen(end\_station); j++) {
         1. if (trains[i].end\_station[j] != end\_station[j]) { // Сравнение символов
            1. is\_equal = false;
            2. break;
         2. }
      2. }
   5. }
   6. if (is\_equal) {
      1. trains\_buffer[selection\_count] = trains[i];
      2. selection\_count++;
   7. }
   8. i++;
5. }
6. print\_train(trains\_buffer, trains\_buffer1, selection\_count, buffer\_count);
7. std::cout << "Количество записей найдено: " << selection\_count << std::endl;
8. 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])

{

1. Train\* trains\_buffer = new Train[count]; // Буфер поездов для выборки
2. int selection\_count = 0; // Количество найденных поездов
3. float down\_time, up\_time; // Пременные времени в часах (дробные)
4. int end\_ch; // Индекс конца цифр часов
5. char hours\_str2[3] = " ", minutes\_str2[3] = " ", hours\_str1[3] = " ", minutes\_str1[3] = " "; // Промежуточные переменные для часов и минут
6. for (int ch = 0; ch < strlen(down\_departure\_time); ch++) {
   1. if (down\_departure\_time[ch] != ':') { // Сравнение с двоеточием
      1. hours\_str1[ch] = down\_departure\_time[ch];
   2. }
   3. else { // Если равны, то записываем индекс
      1. end\_ch = ch + 1;
      2. break;
   4. }
7. }
8. minutes\_str1[0] = down\_departure\_time[end\_ch]; // Запись минут i-ой записи
9. minutes\_str1[1] = down\_departure\_time[end\_ch + 1];
10. down\_time = float(std::stoi(hours\_str1)) + float(std::stoi(minutes\_str1)) / 60.0; // Подсчёт времени i-ой записи
11. for (int ch = 0; ch < strlen(up\_departure\_time); ch++) {
    1. if (up\_departure\_time[ch] != ':') {
       1. hours\_str2[ch] = up\_departure\_time[ch];
    2. }
    3. else {
       1. end\_ch = ch + 1;
       2. break;
    4. }
12. }
13. minutes\_str2[0] = up\_departure\_time[end\_ch];
14. minutes\_str2[1] = up\_departure\_time[end\_ch + 1];
15. up\_time = float(std::stoi(hours\_str2)) + float(std::stoi(minutes\_str2)) / 60.0;
16. int i = 0;
17. while (i < count) {
    1. bool is\_equal = true;
    2. double \_time;
    3. char \_hours\_str[3] = " ", \_minutes\_str[3] = " ";
    4. for (int ch = 0; ch < strlen(trains[i].time\_departure); ch++) {
       1. if (trains[i].time\_departure[ch] != ':') {
          1. \_hours\_str[ch] = trains[i].time\_departure[ch];
       2. }
       3. else {
          1. end\_ch = ch + 1;
          2. break;
       4. }
    5. }
    6. \_minutes\_str[0] = trains[i].time\_departure[end\_ch];
    7. \_minutes\_str[1] = trains[i].time\_departure[end\_ch + 1];
    8. \_time = float(std::stoi(\_hours\_str)) + float(std::stoi(\_minutes\_str)) / 60.0;
    9. if (\_time >= down\_time && \_time <= up\_time) {
       1. trains\_buffer[selection\_count] = trains[i];
       2. selection\_count++;
    10. }
    11. i++;
18. }
19. print\_train(trains\_buffer, trains\_buffer1, selection\_count, buffer\_count);
20. std::cout << "Количество записей найдено: " << selection\_count << std::endl;
21. 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])

{

1. Train\* trains\_buffer = new Train[count]; // Буфер поездов для выборки
2. int selection\_count = 0; // Количество найденных поездов
3. float down\_time, up\_time; // Пременные времени в часах (дробные)
4. int end\_ch; // Индекс конца цифр часов
5. char hours\_str2[3] = " ", minutes\_str2[3] = " ", hours\_str1[3] = " ", minutes\_str1[3] = " "; // Промежуточные переменные для часов и минут
6. for (int ch = 0; ch < strlen(down\_way\_time); ch++) {
   1. if (down\_way\_time[ch] != ':') { // Сравнение с двоеточием
      1. hours\_str1[ch] = down\_way\_time[ch];
   2. }
   3. else { // Если равны, то записываем индекс
      1. end\_ch = ch + 1;
      2. break;
   4. }
7. }
8. minutes\_str1[0] = down\_way\_time[end\_ch]; // Запись минут i-ой записи
9. minutes\_str1[1] = down\_way\_time[end\_ch + 1];
10. down\_time = float(std::stoi(hours\_str1)) + float(std::stoi(minutes\_str1)) / 60.0; // Подсчёт времени i-ой записи
11. for (int ch = 0; ch < strlen(up\_way\_time); ch++) {
    1. if (up\_way\_time[ch] != ':') {
       1. hours\_str2[ch] = up\_way\_time[ch];
    2. }
    3. else {
       1. end\_ch = ch + 1;
       2. break;
    4. }
12. }
13. minutes\_str2[0] = up\_way\_time[end\_ch];
14. minutes\_str2[1] = up\_way\_time[end\_ch + 1];
15. up\_time = float(std::stoi(hours\_str2)) + float(std::stoi(minutes\_str2)) / 60.0;
16. int i = 0;
17. while (i < count) {
    1. bool is\_equal = true;
    2. double \_time;
    3. char \_hours\_str[3] = " ", \_minutes\_str[3] = " ";
    4. for (int ch = 0; ch < strlen(trains[i].time\_way); ch++) {
       1. if (trains[i].time\_way[ch] != ':') {
          1. \_hours\_str[ch] = trains[i].time\_way[ch];
       2. }
       3. else {
          1. end\_ch = ch + 1;
          2. break;
       4. }
    5. }
    6. \_minutes\_str[0] = trains[i].time\_way[end\_ch];
    7. \_minutes\_str[1] = trains[i].time\_way[end\_ch + 1];
    8. \_time = float(std::stoi(\_hours\_str)) + float(std::stoi(\_minutes\_str)) / 60.0;
    9. if (\_time >= down\_time && \_time <= up\_time) {
       1. trains\_buffer[selection\_count] = trains[i];
       2. selection\_count++;
    10. }
    11. i++;
18. }
19. print\_train(trains\_buffer, trains\_buffer1, selection\_count, buffer\_count);
20. std::cout << "Количество записей найдено: " << selection\_count << std::endl;
21. 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\_number, int up\_stop\_number)

{

1. Train\* trains\_buffer = new Train[count]; // Буфер поездов для выборки
2. int selection\_count = 0; // Количество найденных поездов
3. for (int i = 0; auto train: std::span(trains, count)) { // Цикл по записям
   1. int \_number = train.stop\_count;
   2. if (\_number >= down\_stop\_number && \_number <= up\_stop\_number) { // Проверка на диапазон
      1. trains\_buffer[selection\_count] = train;
      2. selection\_count++;
   3. }
   4. i++;
4. }
5. print\_train(trains\_buffer, trains\_buffer1, selection\_count, buffer\_count);
6. std::cout << "Количество записей найдено: " << selection\_count << std::endl;
7. 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)

{

1. int \_number[count];
2. for (int i = 0; auto train: std::span(trains, count)) {
   1. index[i] = i;
   2. \_number[i] = std::stoi(train.number);
   3. i++;
3. }
4. quick\_sort(\_number, 0, count - 1, index);

}

// Сортировка по названию станции

// ======================================================================================

void end\_station\_sort(Train\* trains, int\* index, int count)

{

1. char \_end\_station[count][256];
2. for (int i = 0; auto train: std::span(trains, count)) {
   1. char temp[256];
   2. index[i] = i;
   3. strcpy(temp, train.end\_station);
   4. for (int j = 0; j < strlen(temp); j++) {
      1. temp[j] = tolower(temp[j]);
   5. }
   6. strcpy(\_end\_station[i], temp);
   7. i++;
3. }
4. quick\_sort(\_end\_station, 0, count - 1, index);

}

// Сортировка по времени отправления

// ======================================================================================

void departure\_time\_sort(Train\* trains, int\* index, int count)

{

1. double \_departure\_time[count];
2. for (int i = 0; auto train: std::span(trains, count)) {
   1. int end\_ch;
   2. index[i] = i;
   3. char hours\_str1[3] = " ", minutes\_str1[3] = " "; // Промежуточные переменные для часов и минут
   4. for (int ch = 0; ch < strlen(trains[i].time\_departure); ch++) { // Запись часов i-ой записи
      1. if (train.time\_departure[ch] != ':') { // Сравнение с двоеточием
         1. hours\_str1[ch] = train.time\_departure[ch];
      2. }
      3. else { // Если равны, то записываем индекс
         1. end\_ch = ch + 1;
         2. break;
      4. }
   5. }
   6. minutes\_str1[0] = train.time\_departure[end\_ch]; // Запись минут i-ой записи
   7. minutes\_str1[1] = train.time\_departure[end\_ch + 1];
   8. \_departure\_time[i] = double(std::stoi(hours\_str1)) + double(std::stoi(minutes\_str1)) / 60.0; // Подсчёт времени i-ой записи
   9. i++;
3. }
4. 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)

{

1. double \_way\_time[count];
2. for (int i = 0; auto train: std::span(trains, count)) {
   1. int end\_ch;
   2. index[i] = i;
   3. char hours\_str1[3] = " ", minutes\_str1[3] = " "; // Промежуточные переменные для часов и минут
   4. for (int ch = 0; ch < strlen(train.time\_way); ch++) { // Запись часов i-ой записи
      1. if (train.time\_way[ch] != ':') { // Сравнение с двоеточием
         1. hours\_str1[ch] = train.time\_way[ch];
      2. }
      3. else { // Если равны, то записываем индекс
         1. end\_ch = ch + 1;
         2. break;
      4. }
   5. }
   6. minutes\_str1[0] = train.time\_way[end\_ch]; // Запись минут i-ой записи
   7. minutes\_str1[1] = train.time\_way[end\_ch + 1];
   8. \_way\_time[i] = double(std::stoi(hours\_str1)) + double(std::stoi(minutes\_str1)) / 60.0; // Подсчёт времени i-ой записи
   9. i++;
3. }
4. quick\_sort(\_way\_time, 0, count - 1, index);

}

// Сортировка по количеству остановок

// ======================================================================================

void stop\_count\_sort(Train\* trains, int\* index, int count)

{

1. int \_stop\_count[count];
2. for (int i = 0; auto train: std::span(trains, count)) {
   1. index[i] = i;
   2. \_stop\_count[i] = train.stop\_count;
   3. i++;
3. }
4. quick\_sort(\_stop\_count, 0, count - 1, index);

}

// Функции для быстрой сортировки

// ======================================================================================

void quick\_sort(int\* A, int from, int to, int\* index)

{

1. int x, i, j, temp;
2. if (from >= to) return; // Условие окончания рекурсии
3. i = from; // Рассматриваем элементы с A[from] до A[to]
4. j = to;
5. x = A[(from + to) / 2]; // Выбираем средний элемент
6. while ( i <= j ) {
   1. while (A[i] < x) i++; // Ищем пару для перестановки
   2. while (A[j] > x) j--;
   3. if (i <= j) {
      1. temp = A[i]; A[i] = A[j]; A[j] = temp; // Перестановка
      2. temp = index[i]; index[i] = index[j]; index[j] = temp;
      3. i++; // Двигаемся дальше
      4. j--;
   4. }
7. }
8. quick\_sort(A, from, j, index); // Сортируем левую часть
9. quick\_sort(A, i, to, index); // Сортируем правую часть

}

// ======================================================================================

void quick\_sort(double\* A, int from, int to, int\* index)

{

1. int i, j, index\_temp;
2. double temp, x;
3. if (from >= to) return; // Условие окончания рекурсии
4. i = from; // Рассматриваем элементы с A[from] до A[to]
5. j = to;
6. x = A[(from + to) / 2]; // Выбираем средний элемент
7. while ( i <= j ) {
   1. while (A[i] < x) i++; // Ищем пару для перестановки
   2. while (A[j] > x) j--;
   3. if (i <= j) {
      1. temp = A[i]; A[i] = A[j]; A[j] = temp; // Перестановка
      2. index\_temp = index[i]; index[i] = index[j]; index[j] = index\_temp;
      3. i++; // Двигаемся дальше
      4. j--;
   4. }
8. }
9. quick\_sort(A, from, j, index); // Сортируем левую часть
10. quick\_sort(A, i, to, index); // Сортируем правую часть

}

// ======================================================================================

void quick\_sort(char A[][256], int from, int to, int\* index)

{

1. int i, j, index\_temp;
2. char temp[256], x[256];
3. if (from >= to) return; // Условие окончания рекурсии
4. i = from; // Рассматриваем элементы с A[from] до A[to]
5. j = to;
6. strcpy(x, A[(from + to) / 2]); // Выбираем средний элемент
7. while ( i <= j ) {
   1. while (strcmp(A[i], x) < 0) i++; // Ищем пару для перестановки
   2. while (strcmp(A[j], x) > 0) j--;
   3. if (i <= j) {
      1. strcpy(temp, A[i]); strcpy(A[i], A[j]); strcpy(A[j], temp); // Перестановка
      2. index\_temp = index[i]; index[i] = index[j]; index[j] = index\_temp;
      3. i++; // Двигаемся дальше
      4. j--;
   4. }
8. }
9. quick\_sort(A, from, j, index); // Сортируем левую часть
10. quick\_sort(A, i, to, index); // Сортируем правую часть

}