МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

КУРСОВАЯ РАБОТА (ПРОЕКТ)   
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доц., канд. техн. наук |  |  |  | А.В. Туманова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ |
| РАЗРАБОТКА ПРОГРАММЫ «УЧЕТ УСПЕВАЕМОСТИ СТУДЕНТОВ» |
| по дисциплине: ОСНОВЫ ПРОГРАММИРОВАНИЯ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4134К |  |  |  | Д.О.Маслова |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

ᅠ

Содержание

[1. ПОСТАНОВКА ЗАДАЧИ 3](#_Toc91365175)

[2. ОПИСАНИЕ СТРУКТУР ДАННЫХ 4](#_Toc91365176)

[3. ОПИСАНИЕ ПРОГРАММЫ И СОЗДАННЫХ ФУНКЦИЙ 6](#_Toc91365177)

[4. ОПИСАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА. 12](#_Toc91365178)

[5. РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ 13](#_Toc91365179)

[ЗАКЛЮЧЕНИЕ 22](#_Toc91365180)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 23](#_Toc91365181)

[ПРИЛОЖЕНИЕ 24](#_Toc91365184)

# 1. ПОСТАНОВКА ЗАДАЧИ

1.1. Задачей курсового проекта является разработка программы «расписание поездов» с использованием заданных структур данных, которая позволяет вводить информацию, хранить её в файле, осуществлять поиск, модификацию, сортировку и удаление данных.

1.2. Тип хранимой информации и задание на поиск приводятся в каждом варианте.

Вариант 7.

Предметная область – «Расписание поездов». Данные о маршруте

поезда хранятся в структуре TRAIN, содержащей следующие поля:

 название пункта назначения;

 номер поезда;

 время отправления.

Задание на поиск: найти поезда, направляющиеся в пункт, название

которого введено с клавиатуры.

# 2. ОПИСАНИЕ СТРУКТУР ДАННЫХ

Данные о студенте хранятся в структуре STUDENT.

struct Train {//структура данных

string name;//пункт назначени

string number;//номер поезда

string time;//время

}

Ограничения, накладываемы на выбранные типы данных:

* Пункт назначения

Может быть написан на русском, начинаться с заглавной буквы и состоять только из букв. Буква, идущая после пробела (инициалы) также должна быть заглавной. Строка не должна превышать 30 символов.

* Номер поезда

Формат номера: число от 1000 до 9999. Номер поезда должен состоять только из четырех цифр. Число должно быть написано слитно.

* Время отправления

Время хранится переменной типа string, для сортировки происходит перевод в int. Ввод времени происходит в двух этапах, сначала часы, после ставится автоматически «:», после вводятся минуты. Формат времени: часы от 00 до 23, минуты от 00 до 59.

База данных хранится на диске в виде текстового файла формата .txt.

Пункт назначения, время отправления и время являются уникальными.

Во время работы программы данные хранятся в линейном однонаправленном списке, после выполнения программы изменения, внесенные в список, сохраняются в файл по желанию пользователя. Данные одного поезда, расположенные в одной строке, разделяются символом «|».

# 3. ОПИСАНИЕ ПРОГРАММЫ И СОЗДАННЫХ ФУНКЦИЙ

3.1 Программа реализована на языке С++ в виде консольного приложения. Средой разработки является Visual Studio. В главной функции main() реализовано меню пользователя, в котором каждому действию соответствует определенная цифра. Реализованы следующие функции для работы с данными: добавление, редактирование, удаление записи, сортировка, поиск, загрузка данных из файла и сохранение в текстовый файл, вывод на экран. Описание функций основано на меню пользователя. Функции, напрямую задействованные в меню пользователя, являются основными. А все остальные функции, которые вызываются в ходе выполнения программы, являются вспомогательными.

3.2 **Основные функции.**

**Добавление записи**

void add\_train(Train\* train)

Функция выполняет добавление записи в конец списка. Перед вызовом функции вводятся данные и выполняется проверка формата введенных данных (пункт назначения, номер поезда, время отправления), т.е. вызываются функции проверки. Если введены некорректные данные, пользователю предлагается ввести их повторно. Также выполняется проверка на уникальность поезда. Если пользователь ввел не уникальный поезд, выводится соответствующее сообщение.

**Редактирование записи**

void edit\_info()

Функция выполняет редактирование записи в списке. Пользователь вводит пункт назначения и номер поезда, данные которого хочет изменить. Осуществляется проверка корректности ввода, если ввод некорректен, то пользователю предлагается снова ввести данные. Вызывается функция поиска. Если такого поезда нет в списке, то выводится соответствующее сообщение. Иначе пользователю предоставляется выбор, какое поле он хочет изменить (Пункт назначения/Номер поезда/Время отправления/Выход). Выбор осуществляется вводом числа (1/2/3/0). Если ввод некорректен, то пользователю предлагается повторно ввести число. Далее пользователь вводит выбранное поле. Осуществляется проверка корректности введённого поля. Если поле введено некорректно, то запрашивается повторный ввод. После успешного ввода выбранное поле заменяется на введённое, остальные поля остаются без изменений.

**Удаление записи**

void del\_train()

Функция выполняет удаление записи из списка. Пользователь вводит пункт назначения, данные которого будут удалены. Осуществляется проверка на корректный ввод. Вызывается функция поиска. Если пользователь ввёл поезд, которого нет в списке, будет выведено соответствующее сообщение. Иначе данные о поезде удаляются из списка.

**Поиск записи**

Train\* search()

Функция выполняет поиск записи в списке. Пользователь вводит пункт назначения. Если ввод некорректен, то пользователю предлагается повторно ввести. Далее вводится соответствующее поле. Осуществляется проверка корректности введённого поля. Если поле введено некорректно, пользователю предлагается ввести поле повторно. Осуществляется проверка наличия в списке поезд с выбранным полем, если запись не найдена, то выводится соответствующее сообщение. В результате выводится строка, в которой содержатся все данные о заданном поезде.

**Вывод на экран**

void show();

Функция выводит на экран содержимое списка в виде таблицы.

**Сортировка**

void sort\_point()

Функция выполняет сортировку списка по времени отправления в порядке возрастания. Функция меняет указатель на голову и следующий элемент списка, учитывая время отправления. Время отправления с меньшим первым значением перемещается вверх списка.

**Поиск по заданию**

void search\_task()

Функция находит поезда, чеи пункты назначения совпадают с введенного с клавиатуры. Пользователь вводит пункт назначения. Проходимся по всем элементам списка, до тех пор, пока в списке не останется элементов. Далее выводится результат всех соответствующих поездов. Если таких поездов нет, выводится соответствующее сообщение.

**Сохранение базы данных в файл**

void in\_to\_base();

Функция выполняет сохранение БД в текстовый файл. Перед выполнением происходит проверка на существование файла, в случае его отсутствия будет создан новый пустой файл для записи. Осуществляется проверка открытия файла, если файл открыть не удалось, то выводится соответствующее сообщение. Иначе в файл записываются данные о каждом поезде через разделяющий символ «|» между полями одного поезда и через символ перевода каретки между разными поездами. Запись данных происходит до тех пора, пока не будет достигнут конец списка.

Формат записи данных в файл:

Пункт назначения|Номер\_поезда|Время отправления

Красное Село|8956|13:40

**Загрузить базу данных из файла**

int out\_from\_base();

Функция выполняет выгрузку БД из текстового файла и сохранение в список. Производится проверка открытия файла, если файл открыть не удалось, то выводится соответствующее сообщение. Считывание происходит с помощью функции getline до тех пор, пока не будет достигнут конец файла. Номер поезд переводится из string в int с помощью вызова соответствующей функции, в которой предварительно осуществляется проверка на корректность записанного в файле числа. Пустые строки в файле игнорируются. Осуществляется проверка корректности пункт назначения, номера поезда, их уникальности, и корректность время отправления. Если все данные корректны, то вызывается функция добавления записи. Иначе считывается следующая строка из файла.

3.3 **Вспомогательные функции.**

**Ввод имени и инициалов студента**

string get\_name();

Функция запрашивает пользователя ввести пункт назначения до тех пор, пока ввод не будет корректен. То есть пройдена проверка ввода пункта назначения, и ввод удовлетворяет ограничениям.

**Ввод номера группы**

string get\_number();

Функция запрашивает пользователя ввести номер поезда до тех пор, пока ввод не будет осуществлён корректно. То есть пройдена проверка на корректность ввода номера поезда, и ввод удовлетворяет ограничениям.

**Ввод массива оценок**

string get\_time();

Функция запрашивает пользователя ввести сначала часы, далее минуты до тех пор, пока ввод не будет осуществлён корректно. То есть пройдена проверка корректности ввода времени отправление, и ввод удовлетворяет ограничениям.

**Проверка на корректность ввода пункта назначения**

bool check\_correct\_name(string target);

Функция проверяет длину строки, возвращает результат регулярного выражения, с помощью которого происходит проверка (первая заглавная буква, далее строчные буквы, пробел, заглавная буква, или первая заглавная, далее строчные, или первая заглавная, далее строчные, дефис, заглавная и далее строчные), и ложь в противном случае.

**Проверка на корректность ввода номера поезда**

bool check\_correct\_number(string target);

Функция проверяет корректность строки в соответствии с ограничениями, описанными в пункте описание структур данных. Номер поезда должен состоять ровно из четырёх цифр и не начинаться с 0.

**Проверка на корректность ввода времени отправления**

string get\_time():

Функция проверяет корректность формата времени отправления. Возвращает истину, если оценки – это целые числа в пределах 0-9, и ложь в противном случае.

**Проверка уникальности поезда в списке**

bool check\_uniq(Train\* train);

Функция просматривает весь список и возвращает истину, если пункт назначения и номер поезда уникальный, т.е. не встречается в списке, и возвращает ложь, если не уникальны, т.е. поезд с таким пунктом назначением и номером уже содержится в списке.

# 4. ОПИСАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА.

При запуске программы на экран выводится консольное приложение с меню пользователя, в котором пользователь может увидеть список действий и выбрать нужный ему вариант. При нажатии на клавиатуре на определенную цифру выполняется соответствующая функция. Пользователю будет предлагаться ввод выбора варианта действий до тех пор, пока не будет выбрана команда выхода из программы. Если пользователь введёт недопустимые данные, т.е. не цифру одного из вариантов меню, то ему будет предложено заново их ввести, это будет происходить до первого корректного ввода команды. При выборе определённой команды программа обращается к соответствующей функции.

Варианты, предлагаемые пользователю:

1. Добавить поезд

2. Вывести все

3. Удалить

4. Найти поезда, которые идут в один пункт назначения

5. Редактировать

6. Сохранить

7. Загрузить из базы

8. Сортировка по времени отправления

0. Выход

# 5. РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Результаты тестирования представлены в виде скриншотов с комментариями.

Рисунок 1 – представлена проверка на ввод выбора пункта меню.

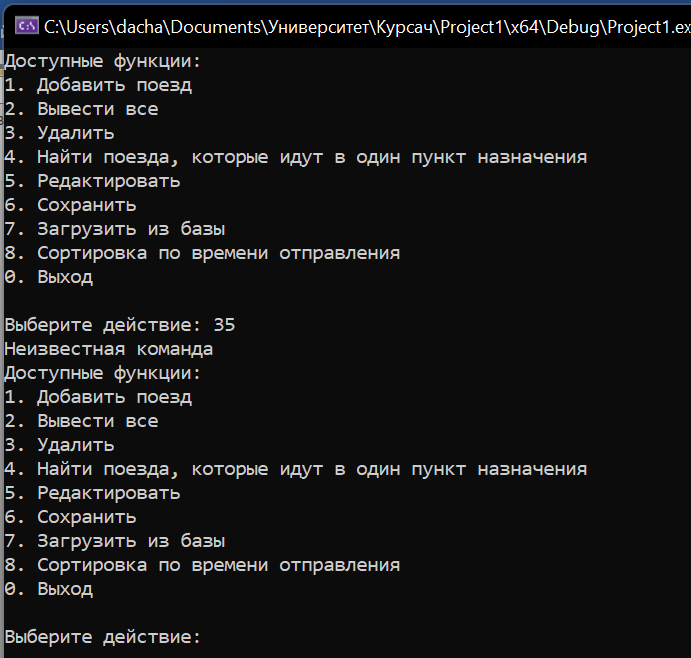
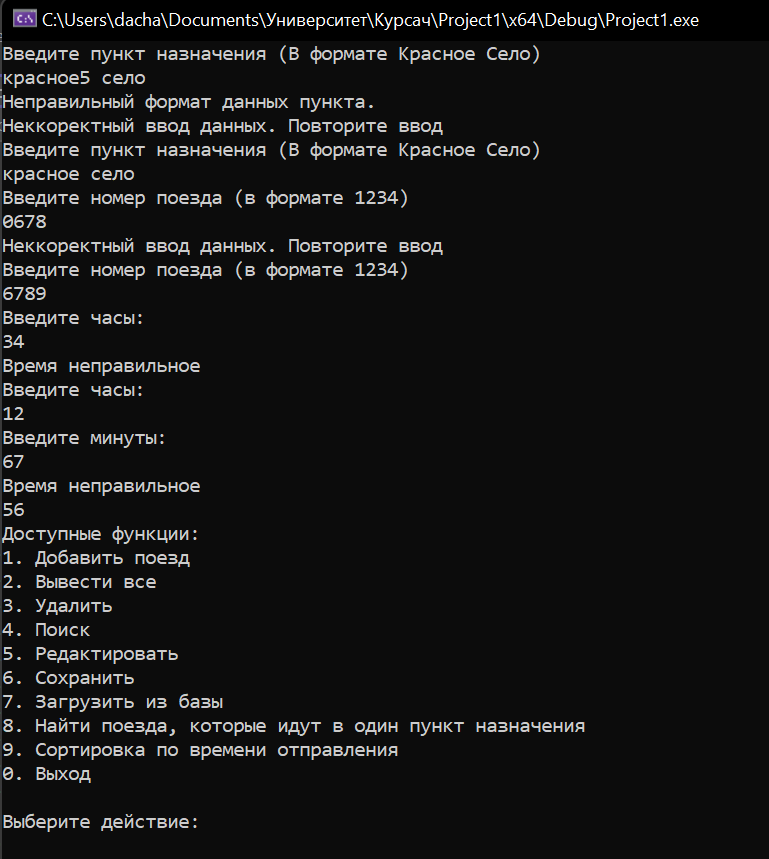


Рисунок 1.

Рисунок 2 – демонстрируется проверка ввода пункта назначения, формата номера поезда и формата времени отправления. Также видно проверку на уникальность по пункту назначения и номеру поезда. Продемонстрирована функция добавления записи в список.



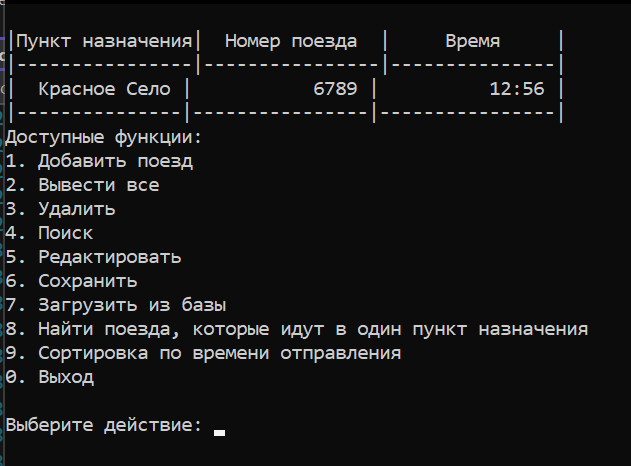


Рисунок 2.

Рисунок 3 – продемонстрирована работа функции вывода списка и функции поиска по пункту назначения.

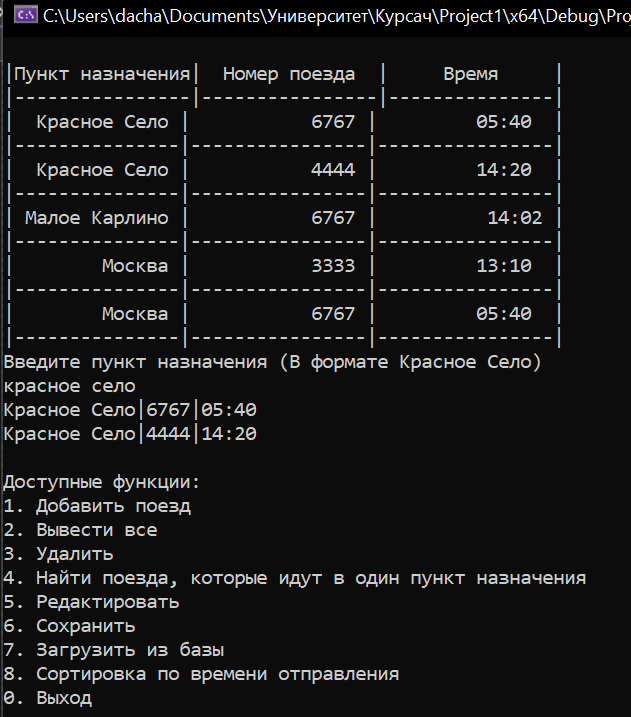


Рисунок 3.

Рисунок 4 – показана работа функции поиска при отсутствии искомого поезда.



Рисунок 4.

Рисунок 5 – сортировка списка по времени отправления.

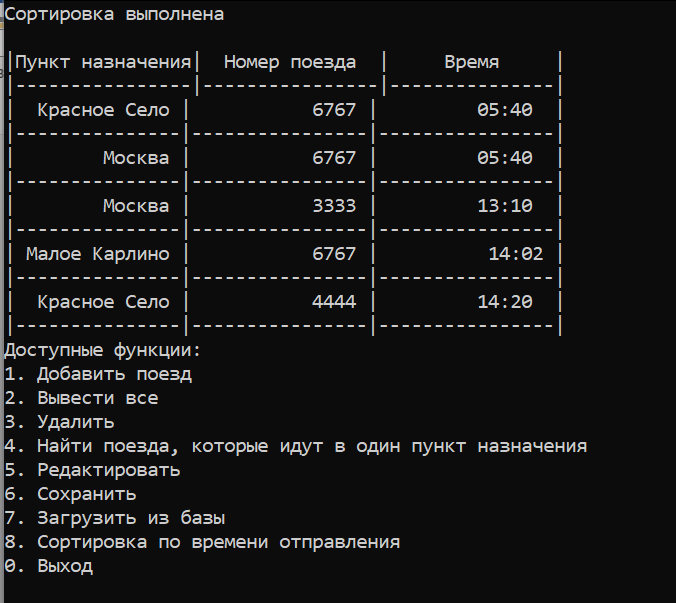
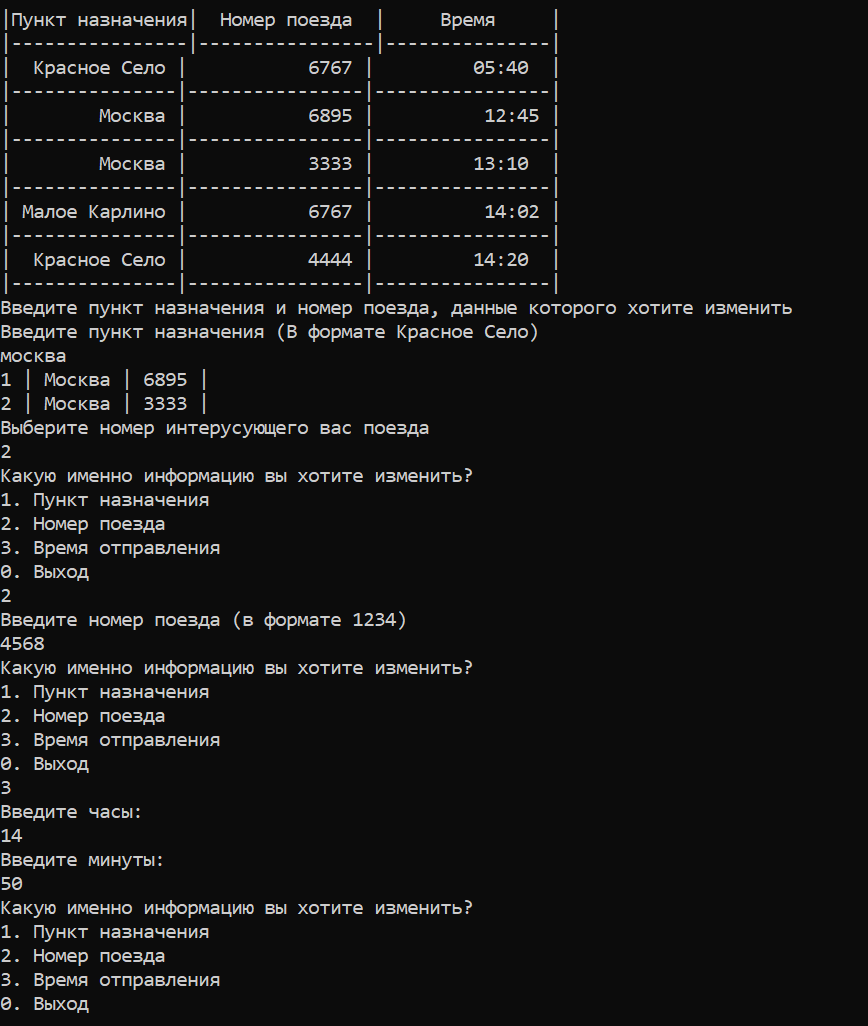


Рисунок 5.

Рисунок 6 – демонстрируется работа функции редактирования записи. Редактируется номер поезда и время отправления.



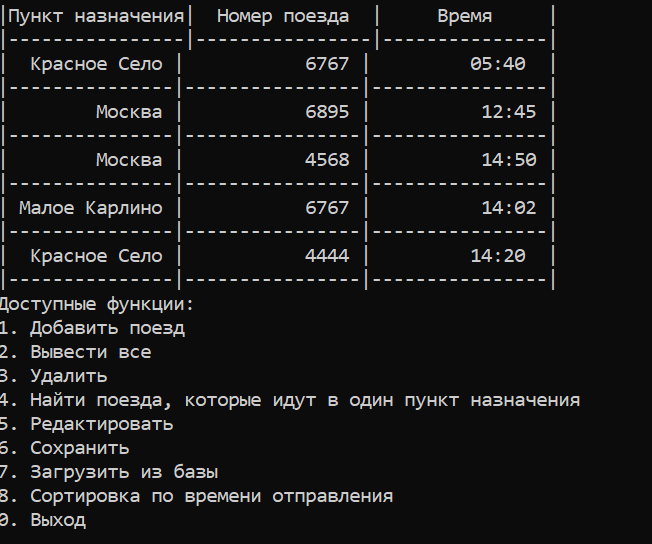


Рисунок 6.

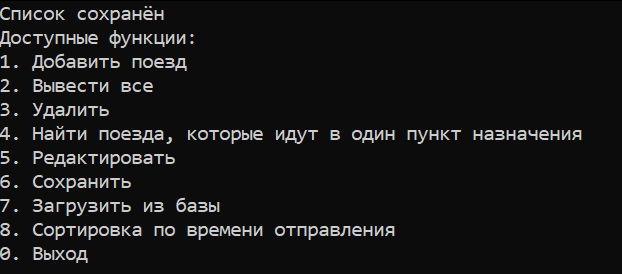
Рисунок 7 – показана работа функции удаления записи.





Рисунок 7.

Рисунок 8 – демонстрируется работа функции загрузки базы данных в файл. Представлен список в виде таблицы в программе и то, как он загрузился в файл.



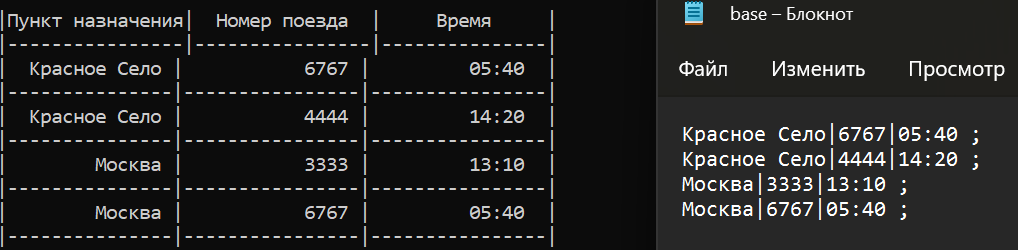


Рисунок 8.

Рисунок 9 – показана работа функции загрузки базы данных из файла. В файле записаны некоторые некорректные данные. В результате работы функции в список записываются только корректные данные, а записи, не удовлетворяющие проверкам, игнорируются.

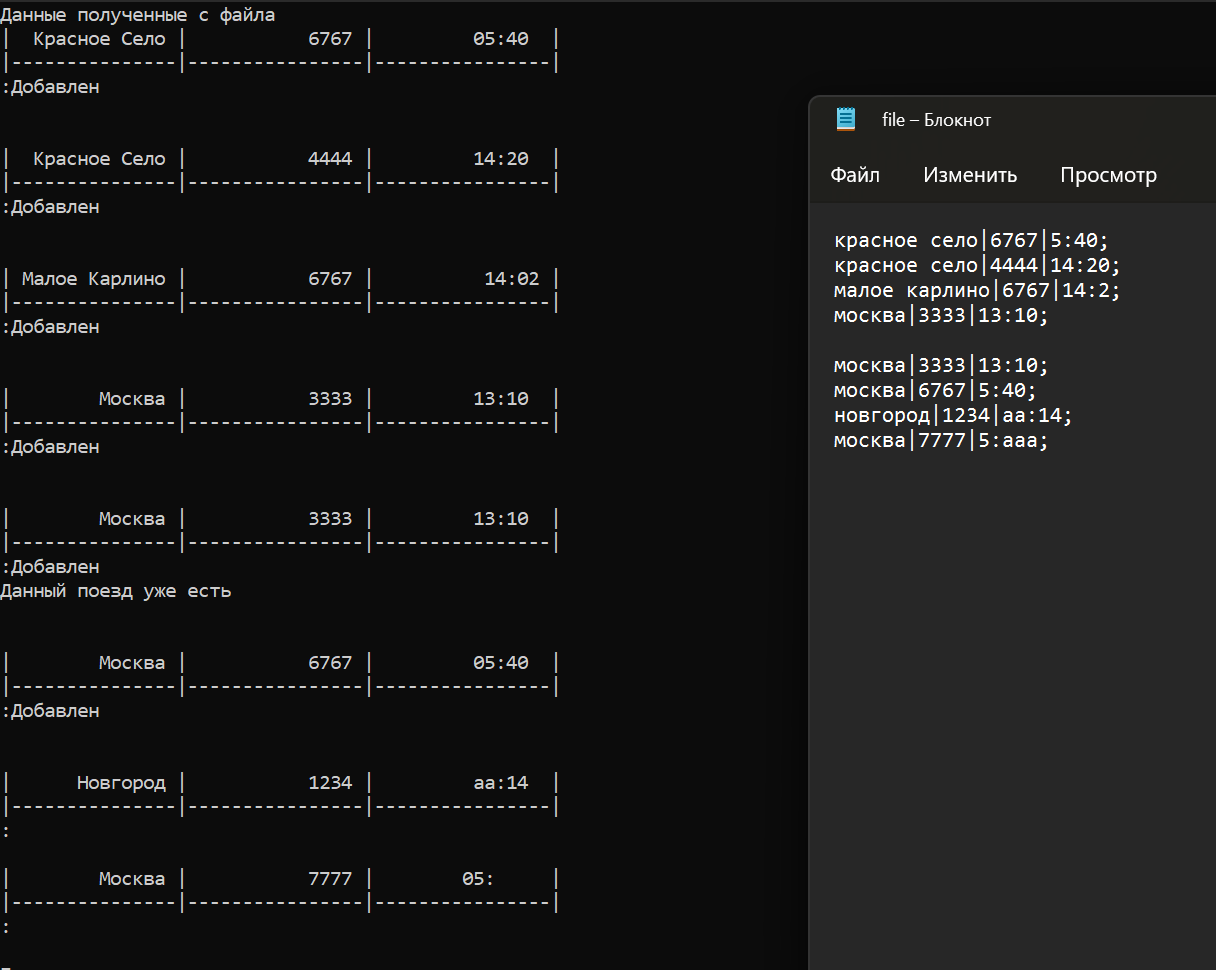




Рисунок 9.

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта были закреплены теоретические и практические знания, полученные на лекционных, лабораторных и практических занятиях по курсу, а также получены навыки разработки, отладки и тестирования программы на алгоритмическом языке программирования. Также закреплены на практике знания по работе с однонаправленными линейными списками. Разработана программа «расписание поездов» на языке С++. Она выполняет все поставленные задачи.

В программе использовались операторы динамического выделения памяти, поэтому были предприняты меры по обнаружению возможных утечек памяти.

Достоинства программы:

* в программе осуществляются различные проверки ввода данных
* программа проста и понятна в использовании
* программа работает корректно и проходит все тесты
* при наличии некорректных значения в БД, они выводится не будут
* предусмотрено предотвращение утечек памяти

Недостатки программы:

* при редактировании и удалении из БД программа не выводит, что изменилось, нужно воспользоваться функцией вывода всех данных
* сортировка осуществляется только по времени

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

# 1. Туманова А.В. основы программирования: Методические указания к выполнению курсового проекта / СПбГУАП. СПб., 2019.

# 2. Ключарёв А.А., Матьяш В.А., Щёкин С.В., Структуры и алгоритмы обработки данных: Учебное пособие / СПБГУАП. СПб., 2004.

3. Демидович, Е.М. Основы алгоритмизации и программирования. Язык

СИ [Текст]: учебное пособие / Е. М. Демидович. - 2-е изд., испр. и доп. - СПб.: БХВ - Петербург, 2008. - 440 с.

# 

# ПРИЛОЖЕНИЕ

**Приложение 1 (текст программы)**

#define \_CRTDBG\_MAP\_ALLOC /

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <clocale>

#include <stdlib.h>

#include <crtdbg.h>

#ifdef \_DEBUG

#ifndef DBG\_NEW

#define DBG\_NEW new ( \_NORMAL\_BLOCK , \_\_FILE\_\_ , \_\_LINE\_\_ )

#define newDBG\_NEW

#endif

#endif

#include <iostream>

#include <iomanip>

#include <cctype>

#include <string>

#include <cstring>

#include <windows.h>

#include <fstream>

#include <regex> //регулярные выражения

using namespace std;

bool check\_length(string n, int length) //Проверка на длину

{

if (n.length() > length || n.length() == 0)

{

cout << "Длина строки должна быть меньше " << length << " символов и больше нуля!" << endl;

return false;

}

return true;

}

int get\_num() {//ввод целого числа с проверкой ввода

int num;

do {

cin >> num;

if (cin.fail()) {

cout << "Неккоретный ввод числа. Повторите ввод\n";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

}

else

break;

} while (true);

return num;

}

struct Train {//структура данных

string name;//пункт назначени

string number;//номер поезда

string time;//время

Train(string Name, string Number, string Time) {//конструктор структуры

name = Name;

number = Number;

time = Time;

}

friend ostream& operator<<(ostream& out, Train\* train) {//функция вывода данных структуры //перегрузка оператора вывода

cout << "| " << setw(13) << train->name << " | " << setw(14) << train->number << " | "

<< setw(14) << train->time << " |" << endl;

cout << "|---------------|----------------|----------------|" << endl;

return out;

}

};

class TrainsData {//класс для хранения и обработки списка структур

private:

struct Node {//структура узла

Train\* train = nullptr;//указатель на структуру

Node\* next = nullptr;//указатель на следующий элемент списка

};

Node\* head;//указатель на начальный элемент

int count;//кол-во элементов в списке

bool check\_correct\_name(string target) {//проверка на корректность ввода пункта назначения

if (!check\_length(target, 30))//проверяем длину

return false;

bool res;

bool res1;

bool res2;

int j = 0;

while (target[j] != '\0') {

j++;

}

char\* enter = new char[j];

for (int i = 0; i < j; i++) {

if (target[i] == '-') {

target[i] = ' ';

res2 = false;

}

}

regex rx(R"(([A-ZА-Яa-zа-я]{1})([a-zа-я]+)\s([A-ZА-Яa-zа-я]{1})([a-zа-я]+))");

regex rx2(R"(([A-ZА-Яa-zа-я]{1})([a-zа-я]+))");//создаем регулярное выражение

res = regex\_match(target.cbegin(), target.cend(), rx);

res1 = regex\_match(target.cbegin(), target.cend(), rx2);//проверяем совпадает ли результат рег. выражения со всей строкой

if (!res && !res1) {

cout << "Неправильный формат данных пункта.\n";

return 0;

}

if (!res1) {

for (int i = 0; i < j; i++) {

if (target[i] == ' ') {

target[i] = '-';

}

}

}

if (res) {

return res;

}

else {

return res1;

}

}

bool check\_correct\_number(string target) {//проверка на корректность ввода номера поезда

if (!check\_length(target, 30))//проверка на длину

return false;

bool res;

regex rx(R"(([\d]{4}))");//создаем регулярное выражение //

res = regex\_match(target.cbegin(), target.cend(), rx);//проверяем совпадает ли результат рег. выражения со всей строкой

if (!res)

cout << "Неправильный формат данных номера поезда.\n";

return res;

}

void check(int& r) {

while (cin.fail())

{

if (cin.get() == '\n') {

break;

}

double x;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

std::cout << " Ошибка : ";

cin >> x;

r = x;

}

}

bool check\_uniq(Train\* train) {//проверка на уникальность элемента в списке

Node\* temp = head;

Train\* s;

for (int i = 0; i < count; i++) {//проходимся по всем элементам списка

s = temp->train;

if (s->name == train->name && s->number == train->number)//сравниваем уникальность на пункта назначения и номера поезда

return false;

temp = temp->next;

}

return true;

}

string get\_name() {//функция для получения пункта с проверкой ввода

string name;//переменная хранящая результат

bool res;

while (true) {

std::cout << "Введите пункт назначения (В формате Красное Село)\n";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(std::cin, name);

if (check\_correct\_name(name)) {//проверка на корректность ввода

name[0] = toupper(name[0]);

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

if (name[i] == ' ' || name[i] == '-') {

name[i + 1] = toupper(name[i + 1]);

}

}

break;

}

else {

cout << "Неккоректный ввод данных. Повторите ввод\n";

}

}

return name;

}

string get\_number() {//функция для получения номера поезда с проверкой ввода

string number;//переменная хранящая результат

while (true) {

std::cout << "Введите номер поезда (в формате 1234)\n";

cin.clear();//очищаем флаги и буфер

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, number);

if (check\_correct\_number(number) && (number[0] != '0')) {//проверка на корректность ввода

break;

}

else {

cout << "Неккоректный ввод данных. Повторите ввод\n";

}

}

return number;

}

string get\_time() {

int min, hour, i;

i = 0;

while (i != 2) {

i = 0;

cout << "Введите часы: " << endl; cin >> hour; check(hour);

if (hour >= 24) {

cout << "Время неправильное" << endl;

continue;

}

else {

i++;

}

cout << "Введите минуты: " << endl; cin >> min; check(min);

while (i != 2) {

if (min >= 60) {

cout << "Время неправильное" << endl; cin >> min; check(min);

continue;

}

else {

i++;

}

}

}

string time;

time = "";

if (hour == 0 || hour == 1 || hour == 2 || hour == 3 || hour == 4 || hour == 5 || hour == 6 || hour == 7 || hour == 8 || hour == 9) {

time = '0' + to\_string(hour) + ':' + to\_string(min);

if (min == 0 || min == 1 || min == 2 || min == 3 || min == 4 || min == 5 || min == 6 || min == 7 || min == 8 || min == 9) {

time = '0' + to\_string(hour) + ':' + '0' + to\_string(min);

}

else

{

time = time = '0' + to\_string(hour) + ':' + to\_string(min);

}

}

else

{

time = to\_string(hour) + ':' + to\_string(min);

if (min == 0 || min == 1 || min == 2 || min == 3 || min == 4 || min == 5 || min == 6 || min == 7 || min == 8 || min == 9) {

time = to\_string(hour) + ':' + '0' + to\_string(min);

}

else

{

time = time = to\_string(hour) + ':' + to\_string(min);

}

}

return time;

}

public:

void sort\_point() {//сортировка поездов по времени

if (count == 0) {

cout << "Список пуст" << endl;

return;

}

cout << "Сортировка выполнена" << endl;

Train\* temp, \* next;

string timestr;

string timestr1;

int time;

int time1;

for (int i = 0; i < count; i++) {//сортируем поезда в массиве по времени

Node\* t = head;

for (int j = 0; j < count - i - 1; j++) {

temp = t->train;

next = t->next->train;

timestr = temp->time;

timestr[2] = '0';

time = stoi(timestr);

timestr1 = next->time;

timestr1[2] = '0';

time1 = stoi(timestr1);

if (time > time1) {//сравниваем время

Train s = \*temp;

\*temp = \*next;

\*next = s;

}

t = t->next;

}

}

}

TrainsData() {//конструктор класса для хранения списка

head = nullptr;

count = 0;

}

Train\* input() {//функция для ввода данных структуры

string name = get\_name();//получаем пункт с проверкой ввода

string number = get\_number();//получаем номер с проверкой ввода

string time = get\_time();//получаем время с проверкой ввода

return new Train(name, number, time);

}

void add\_train(Train\* train) {//функция для добавления поезда в список

if (!check\_uniq(train)) {//проверка на уникальность поезда

cout << "Данный поезд уже есть\n";

delete train;

return;

}

Node\* node = new Node;//создаем новый узел и в него заносим нашу структуру

node->train = train;

if (head == nullptr) {//если нет элементов в списке

head = node;

}

else {//если в списке 1 и более элемент

Node\* temp = head;

for (int i = 0; i < count - 1; i++) {//находим самый последний элемент

temp = temp->next;

}

temp->next = node;//заносим в next последнего элемента новый узел

}

count++;//увеличиваем кол-во элементов

}

void search\_by\_time() {

string coolTime = get\_time();

Train\* temp, \* next;

string timestr;

string timestr1;

Node\* t = head;

int time;

int time1;

int time2;

coolTime[2] = '0';

time2 = stoi(coolTime);

for (int i = 0; i < count; i++) {//сортируем поезда в массиве по времени

temp = t->train;

timestr1 = t->train->time;

timestr1[2] = '0';

time1 = stoi(timestr1);

if (time2 < time1) {

cout << "| " << setw(13) << temp->name << " | " << setw(14) << temp->number << " | "

<< setw(14) << temp->time << " |" << endl;

cout << "|---------------|----------------|----------------|" << endl;

}

t = t->next;

}

}

void del\_train() {//функция для удаления поезда из списка

if (count == 0) {

cout << "Список пуст" << endl;

return;

}

Train\* train = search();//ищем поезд в списке

if (train == nullptr || count == 0) {//если поезд не был найден или список пуст

cout << "Поезд не найден\n";

return;

}

if (count == 1) {//если в списке всего один поезд

delete head;

head = nullptr;

count--;

}

else {//если в списке 1 и более элементов

Node\* prev = nullptr;//указатель хранящий пройденный элемент списка

Node\* temp = head;//устанавливаем указатель на начало списка

for (int i = 0; i < count; i++) {//проходимся по списку

if (train == temp->train) {//если нашли совпадение

if (temp == head) {//если элемент явл-ся первым в списке

head = head->next;

}

else {//если эл-т не явл-ся началом списка

prev->next = temp->next;

}

count--;//уменьшаем кол-во элементов

cout << "Поезд удален\n";

delete temp->train;//освбождаем память

delete temp;

break;

}

prev = temp;//сохраняем пройденный элемент

temp = temp->next;//переставляем ук-ль на след элемент

}

}

}

Train\* search() {//функция для поиска поезд в списке

if (count == 0) {

cout << "Список пуст" << endl;

return 0;

}

string name;

vector<int> arr;

Node\* temp = head;

Train\* w;

int ks{};

cout << ("Введите пункт отправления\n\n");

cin.clear(); //очищаем флаги и буфер

cin.ignore(cin.rdbuf()->in\_avail());

getline(std::cin, name);

w = temp->train;

name[0] = toupper(name[0]);

int it = 1;

while (name[it] != '\0')

{

if (name[it] == ' ')

{

name[++it] = toupper(name[it]);

}

else

{

it++;

}

}

if (name.size() <= w->name.size() && name.size() >= 3 && !(cin.fail()))

{

for (int i = 0; i < count; i++)

{

int j{};

int k{};

int rr{};

w = temp->train;

while (true)

{

if (name.size() < j)

{

break;

}

else if (rr == name.size() / 3)

{

break;

}

if (name[j] == w->name[k])

{

j++;

k++;

}

else

{

rr++;

k++;

j++;

}

if (k == name.size() && rr == 0)

{

std::cout << i << " | " << w->name << " | " << w->number << " | " << endl;

arr.push\_back(i);

ks++;

break;

}

else if (j == name.size() && rr != 0)

{

std::cout << i << " | " << w->name << " | " << w->number << " | " << endl;

arr.push\_back(i);

ks++;

break;

}

}

temp = temp->next;

}

}

else

{

std::cout << "введите символы больше 3 штук\n";

return nullptr;

}

if (ks == 0) {

return nullptr;

}

int i;

bool fl = false;

while (true)

{

cout << "выберите номер поезда " << endl;

cin >> i;

for (int ki = 0; ki < arr.size(); ki++)

{

if (i == arr[ki])

fl = true;

}

if (!fl)

std::cout << "Такого варианта нет\n\n";

else

break;

}

temp = head;

for (int j = 0; j <= i; j++) {

w = temp->train;

if (j == i) {

return w;

}

temp = temp->next;

}

}

void edit\_info() {//функция для редактирования данных о поезде в списке

if (count == 0) {

cout << "Список пуст" << endl;

return;

}

Train\* train = search();//ищем поезд в списке

if (train == nullptr) {//если не нашли поезд

cout << "Поезд не найден\n";

return;

}

string temp1;

Node\* ku = head;

int choice = -1;//номер выбора

string temp;//строковая переменная для хранения полученных данных

string vec;

int k = 0;

while (choice != 0) {

Train s = \*train;//переменная для хранения структуры

cout << "Какую именно информацию вы хотите изменить?\n";

cout << "1. Пункт назначения\n2. Номер поезда\n3. Время отправления\n0. Выход\n";

cin >> choice;

switch (choice) {

case 0:

break;

case 1:

temp = get\_name();

//изменяем пункт назначения на новый

temp1 = train->name;

train->name = temp;

Train\* s;

for (int i = 0; i < count; i++) {//проходимся по всем элементам списка

s = ku->train;

if (s->name == train->name && s->number == train->number) {//сравниваем уникальность на пункт и номер

k++;

}

ku = ku->next;

}

if (k != 1) {

cout << "Пункт назначения не был изменен, данный поезд уже есть" << endl;

train->name = temp1;

}

k = 0;

ku = head;

break;

case 2:

temp = get\_number();

temp1 = train->number;

train->number = temp;

//изменяем номера поезда на новый

for (int i = 0; i < count; i++) {//проходимся по всем элементам списка

s = ku->train;

if (s->name == train->name && s->number == train->number) {//сравниваем уникальность на пункт и номер

k++;

}

ku = ku->next;

}

if (k != 1) {

cout << "Номер поезда не был измене, данный поезд уже есть" << endl;

train->number = temp1;

}

k = 0;

ku = head;

break;

case 3:

vec = get\_time();

train->time = vec;//изменяем время отправления на новое

break;

default:

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

}

}

}

void in\_to\_base() {//загрузка базы данных в файл

if (count == 0) {

cout << "Список пуст" << endl;

return;

}

ofstream base("base.txt"); //создаем файл

if (!base.is\_open()) //проверка на возможность открыть файл для записи

{

cout << "Не удалось записать в файл\n";

return;

}

else

{

base.clear(); //чистим буфер от ошибок

Node\* temp = head;

for (int i = 0; i < count; i++) //проходимся по списку

{

base << temp->train->name << "|" << temp->train->number << "|" << temp->train->time << ";" << endl; //записываем

temp = temp->next; //идем дальше

}

}

cout << "Список сохранён" << endl;

base.close(); //закрываем файл

}

void out\_from\_base() {//загружаем данные из файла в нашу базу данных

ifstream inf("base.txt");

int ind = 0;//индекс для строковой переменной

string str;//строковая переменная, куда мы будем сохранять строки

string name, number;//переменные, куда будем записывать пкнкт и ноер

string time;//массив, куда будем записывать время

Train\* s;//указатель на структуру

string temp;//вспомогательная переменная для хранения данных

if (!inf.is\_open())//проверка на возможность открыть файл для записи

{

cout << "Не удалось открыть файл!" << endl;

return;

}

if (inf.peek() == EOF) {//проверка на пустоту файла

cout << "Файл пуст!\n";

return;

}

cout << "Данные полученные с файла\n";

while (!inf.eof())//пока не конец файла

{

getline(inf, str);//получаем одну строку из файла

if (str.size() == 0)

continue;

int q = 0;

int t = 0;

while (ind < str.size()) {

if (str[ind] == '|') {

q++;

}

if (str[ind] == ';') {

t++;

}

ind++;

}

if (q != 2 || t != 1) {

ind = 0;//обнуляем индекс для строки

temp.clear();

continue;

}

ind = 0;

while (ind < str.size() && str[ind] != '|')//до знака | записываем пункт назначения

{

temp += str[ind];//записываем посимвольно в нашу переменную

ind++;

}

name = temp;//заносим полученное значение в переменную имени

ind++;

name[0] = toupper(name[0]);

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

if (name[i] == ' ') {

name[i + 1] = toupper(name[i + 1]);

}

}

temp.clear();//очищаем перемнную от старых данных для записи новых

while (ind < str.size() && str[ind] != '|')//до знака | записываем номер поезда

{

temp += str[ind];//записываем посимвольно в нашу переменную

ind++;

}

number = temp;//заносим полученное значение в переменную номер

ind++;

temp.clear();//очищаем перемнную от старых данных для записи новых

while (ind < str.size())//до enter записываем время отправления

{

temp += str[ind];//записываем посимвольно в нашу переменную

ind++;

}

time = temp;//заносим полученное значение в переменную имени с преобразованием строки в массив

int i = 0;

int j = 0;

i--;

i = 0;

while (time[i] != ':') {

i++;

}

if (i < 2) {

time = '0' + time;

}

while (time[j] != '\0') {

j++;

}

j--;

if (j < 5) {

time[4] = time[3];

time[3] = '0';

}

j = 3;

char enter;

enter = time[j];

while (enter != '\0') {

enter = time[j];

if (isdigit(time[j])) {

j++;

continue;

}

else {

time[j] = ' ';

j++;

continue;

}

}

s = new Train(name, number, time);

cout << s << ":";

if (check\_correct\_name(name) && check\_correct\_number(number) && isdigit(time[0]) && isdigit(time[1]) && isdigit(time[3]) && isdigit(time[4])) {//если полученные данные корректны, то добавляем в список

cout << "Добавлен\n";

add\_train(s);

}

else

delete s;

cout << "\n\n";

ind = 0;//обнуляем индекс для строки

temp.clear();//очищаем перемнную от старых данных для записи новых

}

cout << "База данных загружена\n" << endl;

inf.close();//закрываем файл

}

void search\_task() {//функция для нахождения поездов, которые идут в один пункт назначения

string average;//переменная для хранения пункта назначения

if (count == 0) {

cout << "Список пуст" << endl;

return;

}

while (true) {

std::cout << "Введите пункт назначения (В формате Красное Село)\n";

cin.clear();//очищаем флаги и буфер

cin.ignore(cin.rdbuf()->in\_avail());

getline(std::cin, average);

if (check\_correct\_name(average)) {//проверка на корректность ввода

average[0] = toupper(average[0]);

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

if (average[i] == ' ') {

average[i + 1] = toupper(average[i + 1]);

}

}

break;

}

else

cout << "Неккоректный ввод данных. Повторите ввод\n";

}

Node\* temp = head;//устанавливаем указатель на начало списка

string destination;//переменная для хранения пункта назначения

Train\* s;//переменная для хранения структуры текущего поезда

string num;

int g = 0;

for (int i = 0; i < count; i++) {//проходимся по списку

s = temp->train;//заносим нашу структуру в переменную s

destination = s->name;//считаем текущий пункт назначения

num = s->number;

if (destination == average) {//если введённый пункт назначения раавен текущему заносим в наш вектор

cout << s->name << "|" << num << "|" << s->time << endl;

g++;

}

temp = temp->next;

}

cout << endl;

if (g == 0) {

cout << "Поезд не найден\n";

}

}

void show() {//функция вывода списка поездов

Node\* temp = head;

if (count == 0) {

cout << "Список пуст" << endl;

return;

}

cout << endl << "|" << setw(0) << "Пункт назначения" << "| " << setw(7) << " Номер поезда " << " | " << setw(10)

<< " Время " << " |" << endl

<< "|----------------|----------------|---------------|" << endl;

for (int i = 0; i < count; i++) //пока не дошли до конца

{

cout << temp->train;

temp = temp->next; //Переставляем указатель на следующий элемент списка

}

}

~TrainsData() {//деструктор списка, который освобождает память из узлов и структур данных

Node\* temp = head;//устанавливаем указатель на начало списка

Node\* prev;//указатель для хранения пройденного элемента

for (int i = 0; i < count; i++) {

prev = temp;

temp = temp->next;

delete prev->train;//освобождаем память из-под узла и структуры

delete prev;

}

}

Node\* getHead()

{

return head;

}

};

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setlocale(0, "");

TrainsData data;

int choice = -1;

Train\* s;

int temp;

while (choice != 0) {

cout << "Доступные функции: " << endl;

cout << "1. Добавить поезд" << endl;

cout << "2. Вывести все" << endl;

cout << "3. Удалить" << endl;

cout << "4. Найти поезда, которые идут в один пункт назначения" << endl;

cout << "5. Редактировать" << endl;

cout << "6. Сохранить" << endl;

cout << "7. Загрузить из базы" << endl;

cout << "8. Сортировка по времени отправления" << endl;

cout << "9. Найти поезда, которые отправляются после вашего времени" << endl;

cout << "0. Выход" << endl;

cout << "\nВыберите действие: ";

choice = get\_num();

switch (choice) {

case 1:

system("cls");

s = data.input();

data.add\_train(s);

break;

case 2:

system("cls");

data.show();

break;

case 3:

system("cls");

data.show();

data.del\_train();

break;

case 4:

system("cls");

data.show();

data.search\_task();

break;

case 5:

system("cls");

data.show();

data.edit\_info();

break;

case 6:

system("cls");

data.in\_to\_base();

break;

case 7:

system("cls");

data.out\_from\_base();

break;

case 8:

system("cls");

data.sort\_point();

data.show();

break;

case 9:

system("cls");

data.show();

data.search\_by\_time();

break;

case 0:

system("cls");

cout << "Выход";

break;

default:

cout << "Неизвестная команда" << endl;

break;

}

}

\_CrtSetReportMode(\_CRT\_WARN, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_WARN, \_CRTDBG\_FILE\_STDOUT);

\_CrtSetReportMode(\_CRT\_ERROR, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_ERROR, \_CRTDBG\_FILE\_STDOUT);

\_CrtSetReportMode(\_CRT\_ASSERT, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_ASSERT, \_CRTDBG\_FILE\_STDOUT);

\_CrtSetDbgFlag(\_CRTDBG\_ALLOC\_MEM\_DF | \_CRTDBG\_LEAK\_CHECK\_DF);

return 0;

}