ГУАП

КАФЕДРА № 14

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ассистент |  |  |  | А.Ю. Петров |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1 |
| по курсу: ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 1042 |  |  |  | Н.В. Корзун |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

1. **Постановка задачи**

Разработать класс Keeper, который реализует контейнер для хранения и обработки

объектов. Класс Keeper должен обеспечивать следующие функции:

▪ добавление и удаление производных объектов абстрактного класса Base (базовый

класс определяется согласно варианту задания);

▪ полное сохранение себя в файле;

▪ полное восстановление себя из файла.

Вариант 9: Класс Грузоперевозчик хранит данные о самолетах, поездах, автомобилях. Для самолета определено: тип, наименование, объем перевозимого груза, габариты, какие города посещает. Для поезда определено: наименование, год выпуска, полный маршрут следования, количество вагонов, объем перевозимого груза. Для автомобиля определено: год выпуска, марка, модель, в какие города осуществляется доставка и для каждого города определено количество часов доставки, объем груза.

1. **Формализация задачи**

Данная программа разбита на пятнадцать файлов

* TP\_lab.cpp - файл содержащий функцию main
* lib.h - файл содержит подключение всех библиотек, как пользовательских так и встроенных, а также макросы подставляемые препроцессором
* menu.h - заголовочный файл для класса Menu
* menu.cpp - файл с реализацией класса Menu
* keeper.h - заголовочный файл для класса Keeper
* keeper.cpp - файл с реализацией класса Keeper
* carrier.h - заголовочный файл для абстрактного класса Carrier
* car.h - заголовочный файл для класса Car
* car.cpp - файл с реализацией класса Car
* plane.h - заголовочный файл для класса Plane
* plane.cpp - файл с реализацией класса Plane
* trane.h - заголовочный файл для класса Trane
* trane.cpp - файл с реализацией класса Trane
* route.h - файл содержащий реализованный контейнер List<> и несколько функций для ввода/вывода в поток
* city.h - файл содержит структуру City, и несколько функций для ввода/вывода в поток

Пользователю доступно два режима работы: ввод данных через консоль (вручную) и из файла. Заранее прописать данные в файл через программу можно чтобы впоследствии их использовать или менять.

Важно: в водимых строках нельзя использовать пробелы и переходы на новую строку. А в вводимые числа должны быть не отрицательны. Нельзя вводить строки, когда требуется ввести число.

В программе реализована работа с динамической памятью. В программе присутствует обработка исключений, также Подстановка макросов препроцессором.

Файлы разделены на .h и.cpp.

Для классов определены методы для просмотра и установки значений (set и get). Также созданы Конструкторы и деструкторы, вызов которых сопровождается извещением в консоль.

Через меню есть доступ ко всем членам объектов-наследников.

1. **Структуры, функции, классы**

В программе определено две структуры: Elem – структура для реализации односвязного списка необходимого для хранения элементов дочерних классов; City – структура содержащая информацию о городе.

В программе присутствуют перегруженные функции потокового ввода/вывода, необходимые для записи в файл и чтения из него. Также определены функции для форматированного ввода/вывода для класса List<> и структуры City.

Классы Plane, Trane, Car – являются дочерними классами абстрактного класса Carrier. Наследование публичное. Также есть класс Menu в котором происходит взаимодействие пользователя с программой. Класс Keeper реализует поставленную задачу – хранение наследников базового класса, путем сохранения их в односвязный список. Класс List<> реализует удобное хранение маршрута, состоящего из структур City.

В каждом дочернем классе переопределены функции *show*, *saving*, *rewrite*, *redact* определены конструкторы и деструкторы.

Таблица 1. Описание всех классов: наследование

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Имя | Назначение | Наследование, указан родительский класс | Тип наследования |
| 1 | Carrier | Абстрактный класс для реализации функционала «Грузоперевозчик» | - | - |
| 2 | List | Класс реализующий коллекцию List, нужен чтобы оперировать маршрутом | - | - |
| 3 | Menu | Класс необходимый для реализации пользовательского управления | - | - |
| 4 | Keeper | Класс для хранения и сохранения элементов дочерних классов | - | - |
| 5 | Plane | Дочерний класс самолет | Carrier | Простое Public |
| 6 | Car | Дочерний класс машина | Carrier | Простое Public |
| 7 | Trane | Дочерний класс поезд | Carrier | Простое Public |

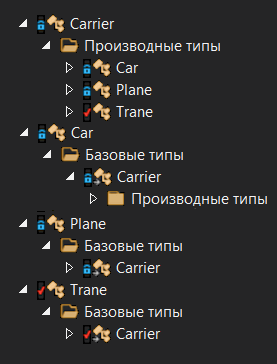


Рисунок 1. Иерархия наследования

Иерархию связи классов – композицию опишем с помощью таблицы.

Таблица 1. Описание всех классов: композиция

|  |  |  |  |
| --- | --- | --- | --- |
| № | Класс | Поле класса | Назначение |
| 1 | Plane | List<City> cities | какие города посещает |
| 2 | Trane | List<City> route; | полный маршрут следования |
| 3 | Car | List<City> cities; | в какие города  осуществляется доставка |
| 4 | Menu | Keeper list; | Экземпляр хранителя |
| 4 | Menu | Carrier\* transports; | Указатель на базовый класс, создаваемых элементов и обрабатываемых элементов-экземпляров дочерних классов |
| 5 | List | Node | Вложенный класс, помогающий в реализации односвязного списка |
| 6 | Keeper | Elem\* head | Указатель на структуры для реализации односвязного списка |

1. **Исходный код**

Файл TP\_lab1.cpp

#include"lib.h"

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

Menu menu;

menu.main\_menu();

return 0;

}

Файл lib.h

#pragma once

#include <iostream>

#include <string>

#include <fstream>

#define IO\_FILENAME "data.txt"

#define ERROR\_MESSAGE\_OPEN\_FILE\_ERROR "ERR! Error opening file"

#define ERROR\_MESSAGE\_INCORRECT\_INDEX "ERR! Incorrect index"

#define ERROR\_MESSAGE\_EMPTY "ERR! Empty"

#define ERROR\_MESSAGE\_EXCEPTION\_THROW "An exception is thrown"

#define ERROR\_MESSAGE\_EXCEPTION\_HANDLED "Exception handled"

#define RAILWAY\_CARRIAGE\_1 " /##\\ /##\\ "

#define RAILWAY\_CARRIAGE\_2 " ########### "

#define RAILWAY\_CARRIAGE\_3 " ## ## ## ## "

#define RAILWAY\_CARRIAGE\_4 "-###########-"

#define RAILWAY\_CARRIAGE\_5 " ©© ©© "

#include "city.h"

#include "route.h"

#include "carrier.h"

#include "car.h"

#include "plane.h"

#include "trane.h"

#include "keeper.h"

#include "menu.h"

Файл menu.h

#pragma once

#include "lib.h"

class Menu {

protected:

Keeper list;

Carrier\* transports;

int flag\_draw\_train = 0;

int exit = 1;

public:

void main\_menu();

int show\_options();

void show\_all\_transport();

void change\_data();

void add\_new();

void save\_to\_file();

void load\_from\_file();

void delete\_transport();

void exit\_from\_programm();

};

Файл menu.cpp

#include "lib.h"

using namespace std;

void Menu::main\_menu()

{

while (exit == 1) {

switch (show\_options())

{

case 1:

show\_all\_transport();

break;

case 2:

change\_data();

break;

case 3:

add\_new();

break;

case 4:

save\_to\_file();

break;

case 5:

load\_from\_file();

break;

case 6:

delete\_transport();

break;

case 7:

exit\_from\_programm();

break;

case 666:

cout << "draw tranes ON\n";

flag\_draw\_train = 1;

break;

}

}

}

int Menu::show\_options()

{

int c1;

cout << " Select action:" << endl;

cout << "1 - Show all transport" << endl;

cout << "2 - Edit data" << endl;

cout << "3 - Add transport" << endl;

cout << "4 - Save in file: \"" << IO\_FILENAME<<"\"" << endl;

cout << "5 - Load from file: \"" << IO\_FILENAME << "\"" << endl;

cout << "6 - Delete transport" << endl;

cout << "666 - For secret options" << endl;

cout << "7 - Exit" << endl;

cin >> c1;

return c1;

}

void Menu::show\_all\_transport()

{

try

{

if (list.get\_size() == 0) {

throw ERROR\_MESSAGE\_EMPTY;

}

for (int i = 0; i < list.get\_size(); i++) {

list[i]->show();

}

}

catch (const char\* er)

{

cout << er << endl;

cout << ERROR\_MESSAGE\_EXCEPTION\_HANDLED << endl;

}

}

void Menu::change\_data()

{

//#TODO

int c2;

try

{

cout << "What transport do you want to change?" << endl;

cout << "Input number from 1 to " << list.get\_size() << endl;

cin >> c2;

if (c2<1 || c2>list.get\_size()) {

throw ERROR\_MESSAGE\_INCORRECT\_INDEX;

}

list[c2 - 1]->show();

cout << " Select action:" << endl;

cout << "1 - Change all options" << endl;

cout << "2 - Change route only" << endl;

int par;

cin >> par;

switch (par) {

case 1:

cout << "Сhange: " << endl;

list[c2 - 1]->rewrite();

break;

case 2:

List<City> r;

cin >> r;

list[c2 - 1]->redact(r);

break;

}

cout << "Data changed." << endl;

}

catch (const char\* err)

{

cout << err << endl;

cout << ERROR\_MESSAGE\_EXCEPTION\_HANDLED << endl;

}

}

void Menu::add\_new()

{

int c3;

cout << "What transport needs to be added?" << endl;

cout << "1 - Car" << endl;

cout << "2 - Plane" << endl;

cout << "3 - Trane" << endl;

cin >> c3;

switch (c3) {

case 1:

Car \* car;

car = new Car;

transports = car;

car->rewrite();

list.insert(transports);

break;

case 2:

Plane \* plane;

plane = new Plane;

transports = plane;

plane->rewrite();

list.insert(transports);

break;

case 3:

Trane \* trane;

trane = new Trane;

transports = trane;

trane->rewrite();

if(flag\_draw\_train)

trane->draw();

list.insert(transports);

break;

default:

break;

}

cout << "New transport has been added!" << endl;

}

void Menu::save\_to\_file()

{

try

{

if (list.get\_size() == 0) {

throw ERROR\_MESSAGE\_EMPTY;

}

list.save();

cout << "Data saved into file \""<< IO\_FILENAME <<"\"" << endl;

}

catch (...)

{

cout << ERROR\_MESSAGE\_EXCEPTION\_THROW << endl;

cout << ERROR\_MESSAGE\_EXCEPTION\_HANDLED << endl;

}

}

void Menu::load\_from\_file()

{

list.load();

cout << "Data extracted from file \"" << IO\_FILENAME << "\"" << endl;

}

void Menu::delete\_transport()

{

int c4;

try

{

if (list.get\_size() == 0) {

throw ERROR\_MESSAGE\_EMPTY;

}

}

catch (...)

{

cout << ERROR\_MESSAGE\_EXCEPTION\_THROW << endl;

cout << ERROR\_MESSAGE\_EXCEPTION\_HANDLED << endl;

}

cout << "What transport needs to be deleted?" << endl;

cout << "Input number from 1 to " << list.get\_size() << endl;

try

{

cin >> c4;

if (c4<1 || c4>list.get\_size())

{

throw ERROR\_MESSAGE\_INCORRECT\_INDEX;

}

list.remove(list.get\_size() - c4);

cout << "Selected transport removed" << endl;

}

catch (...)

{

cout << ERROR\_MESSAGE\_EXCEPTION\_THROW << endl;

cout << ERROR\_MESSAGE\_EXCEPTION\_HANDLED << endl;

}

}

void Menu::exit\_from\_programm()

{

exit = 0;

}

Файл keeper.h

#pragma once

#include "lib.h"

using namespace std;

class Keeper {

private:

int size;

Elem\* head = nullptr;

public:

Keeper();

Keeper(int size);

~Keeper();

int get\_size();

Carrier\* operator[] (const int index);

void insert(Carrier\* n\_data);

void remove(int index);

void save();

void load();

};

Файл keeper.cpp

#include "lib.h"

using namespace std;

Keeper::Keeper()

{

size = 0;

head = nullptr;

}

Keeper::Keeper(int size) :head(nullptr)

{

this->size = size;

}

Keeper::~Keeper() {

if (head != NULL)

{

Elem\* buffer;

while (head->next != NULL)

{

buffer = head;

head = head->next;

buffer->data->~Carrier();

delete(buffer);

}

head->data->~Carrier();

delete(head);

}

}

int Keeper::get\_size()

{

return size;

}

Carrier\* Keeper::operator[] (const int index)

{

if ((index >= size) || (index < 0))

{

throw ERROR\_MESSAGE\_INCORRECT\_INDEX;

}

else

{

Elem\* buffer = head;

for (int i = 0; i < size - index - 1; i++)

{

buffer = buffer->next;

}

return (buffer->data);

}

}

void Keeper::insert(Carrier\* n\_data)

{

Elem\* tmp;

tmp = new Elem;

if (size == 0) {

tmp->data = n\_data;

tmp->next = 0;

++size;

}

else {

tmp->data = n\_data;

tmp->next = head;

++size;

}

head = tmp;

}

void Keeper::remove(int index)

{

if (size == 0) {

throw ERROR\_MESSAGE\_EMPTY;

}

int cnt = 0;

while (cnt < index - 1)

{

head = head->next;

cnt++;

}

Elem\* prev = head;

prev->next->data->~Carrier();

prev->next->next;

delete(prev->next);

--size;

}

void Keeper::save()

{

ofstream fout;

string initfile = IO\_FILENAME;

fout.open(initfile, ios\_base::app);

if (!fout)

{

throw ERROR\_MESSAGE\_OPEN\_FILE\_ERROR;

system("pause");

exit(1);

}

else

{

fout << size << endl;

fout.close();

}

Elem\* buf = head;

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

buf->data->saving();

buf = buf->next;

}

}

void Keeper::load()

{

setlocale(LC\_ALL, "Russian");

if (size != 0)

{

Elem\* buffer;

while (head->next != NULL)

{

buffer = head;

head = head->next;

buffer->data->~Carrier();

delete(head);

}

head->data->~Carrier();

delete(head);

}

ifstream fin;

string initfile = IO\_FILENAME;

Carrier\* transports;

int fsize;// для считывания количества объектов в файле

int num\_type;//1- машина, 2- самолет, 3= поезд

string plane\_type;

string plane\_name;

double plane\_cargo\_volume;

double plane\_cargo\_size;

List<City> plane\_cities;

string car\_brand;

string car\_model;

uint32\_t car\_year\_of\_issue;

List<City> car\_cities;

string trane\_name;

uint32\_t trane\_year\_of\_issue;

uint32\_t trane\_count\_railway\_carriage;

double trane\_cargo\_size;

List<City> trane\_route;

fin.open(initfile);

if (!fin) {

throw ERROR\_MESSAGE\_OPEN\_FILE\_ERROR;

system("pause");

exit(1);

}

fin >> fsize;

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

fin >> num\_type;

//fin.ignore(32767, '\n');

if (num\_type == 1) {

fin >> car\_brand >> car\_model >> car\_year\_of\_issue >> car\_cities;

Car\* car;

car = new Car;

car->set\_brand(car\_brand);

car->set\_model(car\_model);

car->set\_year\_of\_issue(car\_year\_of\_issue);

car->set\_cities(car\_cities);

transports = car;

insert(transports);

}

if (num\_type == 2) {

fin >> plane\_type >> plane\_name >> plane\_cargo\_volume >> plane\_cargo\_size>> plane\_cities;

Plane\* plane;

plane = new Plane;

plane->set\_type(plane\_type);

plane->set\_name(plane\_name);

plane->set\_cargo\_volume(plane\_cargo\_volume);

plane->set\_cargo\_size(plane\_cargo\_size);

plane->set\_cities(plane\_cities);

transports = plane;

insert(transports);

}

if (num\_type == 3)

{

fin >> trane\_name >> trane\_year\_of\_issue >> trane\_count\_railway\_carriage >> trane\_cargo\_size >> trane\_route;

Trane\* trane;

trane = new Trane;

trane->set\_name(trane\_name);

trane->set\_year\_of\_issue(trane\_year\_of\_issue);

trane->set\_count\_railway\_carriage(trane\_count\_railway\_carriage);

trane->set\_cargo\_size(trane\_cargo\_size);

trane->set\_route(trane\_route);

transports = trane;

insert(transports);

}

fin.ignore(32767, '\n');

}

}

Файл carrier.h

#pragma once

#include "lib.h"

using namespace std;

class Carrier

{

public:

Carrier() {};

virtual ~Carrier() {};

virtual void show() = 0;

virtual void saving() = 0;

virtual void rewrite() = 0;

virtual void redact(List<City> route) = 0;

};

typedef struct Elem

{

Carrier\* data;

Elem\* next;

} Elem;

Файл car.h

#pragma once

#include "lib.h"

using namespace std;

class Car : public Carrier

{

public:

Car();

Car(string model);

Car(const Car& car\_copy);

~Car();

void set\_brand(string brand);

void set\_model(string model);

void set\_year\_of\_issue(uint32\_t year\_of\_issue);

void set\_cities(List<City> route);

string get\_brand() const;

string get\_model() const;

uint32\_t get\_year\_of\_issue() const;

List<City> get\_cities() const;

virtual void show() override;

virtual void saving() override;

virtual void rewrite() override;

void redact(List<City> route) override;

private:

string brand;

string model;

uint32\_t year\_of\_issue;

List<City> cities;

};

Файл car.cpp

#include "lib.h"

Car::Car() {

cout << "<Constructor Car without parameters was called>" << endl;

brand = " ";

model = " ";

year\_of\_issue = 0;

cities;

}

Car::Car(string model) {

cout << "<Constructor Car with parameters was called>" << endl;

this->model = model;

}

Car::Car(const Car& car\_copy)

{

cout << "<Copy constructor Car was called>" << endl;

brand = car\_copy.brand;

model = car\_copy.model;

year\_of\_issue = car\_copy.year\_of\_issue;

cities = car\_copy.cities;

}

Car::~Car()

{

cout << "<Destructor Car was called>" << endl;

}

void Car::set\_brand(string brand) {

this->brand = brand;

}

void Car::set\_model(string model) {

this->model = model;

}

void Car::set\_year\_of\_issue(uint32\_t year\_of\_issue) {

this->year\_of\_issue = year\_of\_issue;

}

void Car::set\_cities(List<City> route) {

this->cities = route;

}

string Car::get\_brand() const {

return brand;

}

string Car::get\_model() const {

return model;

}

uint32\_t Car::get\_year\_of\_issue() const {

return year\_of\_issue;

}

List<City> Car::get\_cities() const {

return cities;

}

void Car::show() {

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

cout << "Car" << endl;

cout << " Model: "<<model<<endl;

cout << " Brand: "<<brand << endl;

cout << " Year of issue: "<<year\_of\_issue << endl;

Route\_show(cout, cities);

}

void Car::saving() {

ofstream fout;

string initfile = IO\_FILENAME;

fout.open(initfile, ios\_base::app);

if (!fout)

{

throw ERROR\_MESSAGE\_OPEN\_FILE\_ERROR;

system("pause");

exit(1);

}

else

{

fout << "1" << endl << model << endl << brand << endl << year\_of\_issue << endl << cities << endl;

fout.close();

}

}

void Car::rewrite()

{

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

cout << "Car" << endl;

cout << " Input model: " << endl;

cin >> model;

cout << " Input brand: " << endl;

cin >> brand;

cout << " Input year of issue: " << endl;

cin >> year\_of\_issue;

cout << " Input cities: " << endl;

Route\_input(cin, cities);

}

void Car::redact(List<City> route) {

this->cities = route;

}

Файл plane.h

#pragma once

#include "lib.h"

using namespace std;

class Plane : public Carrier

{

public:

Plane();

Plane(string type);

Plane(const Plane& plane\_copy);

~Plane();

void set\_type(string type);

void set\_name(string name);

void set\_cargo\_volume(double cargo\_volume);

void set\_cargo\_size(double cargo\_size);

void set\_cities(List<City> route);

string get\_type() const;

string get\_name() const;

double get\_cargo\_volume() const;

double get\_cargo\_size() const;

List<City> get\_cities() const;

virtual void show() override;

virtual void saving() override;

virtual void rewrite() override;

void redact(List<City> route) override;

private:

string type;

string name;

double cargo\_volume;

double cargo\_size;

List<City> cities;

};

Файл plane.cpp

#include "lib.h"

using namespace std;

Plane::Plane() {

type = " ";

name = " ";

cargo\_volume = 0;

cargo\_size = 0;

cities;

cout << "<Constructor Plane without parameters was called>" << endl;

}

Plane::Plane(string type) {

this->type = type;

cout << "<Constructor Plane with parameters was called>" << endl;

}

Plane::Plane(const Plane& plane\_copy)

{

cout << "<Copy constructor Plane was called>" << endl;

type = plane\_copy.type;

name = plane\_copy.name;

cargo\_volume = plane\_copy.cargo\_volume;

cargo\_size = plane\_copy.cargo\_size;

cities = plane\_copy.cities;

}

Plane::~Plane()

{

cout << "<Destructor Plane was called>" << endl;

}

void Plane::set\_type(string type) {

this->type = type;

}

void Plane::set\_name(string name) {

this->name = name;

}

void Plane::set\_cargo\_volume(double cargo\_volume) {

this->cargo\_volume = cargo\_volume;

}

void Plane::set\_cargo\_size(double cargo\_size) {

this->cargo\_size = cargo\_size;

}

void Plane::set\_cities(List<City> route) {

this->cities = route;

}

string Plane::get\_type() const{

return type;

}

string Plane::get\_name() const {

return name;

}

double Plane::get\_cargo\_volume() const {

return cargo\_volume;

}

double Plane::get\_cargo\_size() const {

return cargo\_size;

}

List<City> Plane::get\_cities() const {

return cities;

}

void Plane::show() {

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

cout << "Plane" << endl;

cout << " Type: " << type << endl;

cout << " Name: " << name << endl;

cout << " Cargo volume: " << cargo\_volume << endl;

cout << " Cargo size: " << cargo\_size << endl;

Route\_show(cout, cities);

}

void Plane::saving() {

ofstream fout;

string initfile = IO\_FILENAME;

fout.open(initfile, ios\_base::app);

if (!fout)

{

throw ERROR\_MESSAGE\_OPEN\_FILE\_ERROR;

system("pause");

exit(1);

}

else

{

fout << "2" << endl << type << endl << name << endl << cargo\_volume << endl << cargo\_size << endl << cities << endl;

fout.close();

}

}

void Plane::rewrite()

{

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

cout << "Plane" << endl;

cout << " Input type: " << endl;

cin >> type;

cout << " Input name: " << endl;

cin >> name;

cout << " Input cargo volume: " << endl;

cin >> cargo\_volume;

cout << " Input cargo size: " << endl;

cin >> cargo\_size;

cout << " Input cities: " << endl;

Route\_input(cin, cities);

//cin >> cities;

}

void Plane::redact(List<City> route) {

this->cities = route;

}

Файл trane.h

#pragma once

#include "lib.h"

using namespace std;

class Trane : public Carrier

{

public:

Trane();

Trane(string type);

Trane(const Trane& trane\_copy);

~Trane();

void set\_name(string name);

void set\_year\_of\_issue(uint32\_t year\_of\_issue);

void set\_count\_railway\_carriage(uint32\_t count\_railway\_carriage);

void set\_cargo\_size(double cargo\_size);

void set\_route(List<City> route);

string get\_name() const;

uint32\_t get\_year\_of\_issue() const;

uint32\_t get\_count\_railway\_carriage() const;

double get\_cargo\_size() const;

List<City> get\_route() const;

virtual void show() override;

virtual void saving() override;

virtual void rewrite() override;

void redact(List<City> route) override;

void draw();

private:

string name;

uint32\_t year\_of\_issue;

uint32\_t count\_railway\_carriage;

double cargo\_size;

List<City> route;

};

Файл trane.cpp

#include "lib.h"

using namespace std;

Trane::Trane() {

name = " ";

year\_of\_issue = 0;

count\_railway\_carriage = 0;

cargo\_size = 0;

route;

cout << "<Constructor Trane without parameters was called>" << endl;

}

Trane::Trane(string name) {

this->name = name;

cout << "<Constructor Trane with parameters was called>" << endl;

}

Trane::Trane(const Trane& trane\_copy)

{

cout << "<Copy constructor Trane was called>" << endl;

name = trane\_copy.name;

year\_of\_issue = trane\_copy.year\_of\_issue;

count\_railway\_carriage = trane\_copy.count\_railway\_carriage;

cargo\_size = trane\_copy.cargo\_size;

route = trane\_copy.route;

}

Trane::~Trane()

{

cout << "<Destructor Trane was called>" << endl;

}

void Trane::set\_name(string name) {

this->name = name;

}

void Trane::set\_year\_of\_issue(uint32\_t year\_of\_issue) {

this->year\_of\_issue = year\_of\_issue;

}

void Trane::set\_count\_railway\_carriage(uint32\_t count\_railway\_carriage) {

this->count\_railway\_carriage = count\_railway\_carriage;

}

void Trane::set\_cargo\_size(double cargo\_size) {

this->cargo\_size = cargo\_size;

}

void Trane::set\_route(List<City> route) {

this->route = route;

}

string Trane::get\_name() const {

return name;

}

uint32\_t Trane::get\_year\_of\_issue() const {

return year\_of\_issue;

}

uint32\_t Trane::get\_count\_railway\_carriage() const {

return count\_railway\_carriage;

}

double Trane::get\_cargo\_size() const {

return cargo\_size;

}

List<City> Trane::get\_route() const {

return route;

}

void Trane::show() {

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

cout << "Trane" << endl;

cout << " Name: " << name << endl;

cout << " Year of issue: " << year\_of\_issue << endl;

cout << " Count railway carriage: " << count\_railway\_carriage << endl;

cout << " Cargo size: " << cargo\_size << endl;

Route\_show(cout, route);

}

void Trane::saving() {

ofstream fout;

string initfile = IO\_FILENAME;

fout.open(initfile, ios\_base::app);

if (!fout)

{

throw ERROR\_MESSAGE\_OPEN\_FILE\_ERROR;

system("pause");

exit(1);

}

else

{

fout << "3" << endl << name << endl << year\_of\_issue << endl << count\_railway\_carriage << endl << cargo\_size << endl << route << endl;

fout.close();

}

}

void Trane::rewrite()

{

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

cout << "Trane" << endl;

cout << " Input name: " << endl;

cin >> name;

cout << " Input year of issue: " << endl;

cin >> year\_of\_issue;

cout << " Input count railway carriage: " << endl;

cin >> count\_railway\_carriage;

cout << " Input cargo size: " << endl;

cin >> cargo\_size;

cout << " Input route: " << endl;

Route\_input(cin, route);

}

void Trane::redact(List<City> route) {

this->route = route;

}

void Trane::draw()

{

cout << " # ";

for (uint32\_t i = 0; i < count\_railway\_carriage; i++)

{

cout << RAILWAY\_CARRIAGE\_1;

}

cout<<endl;

cout << " # ###### ";

for (uint32\_t i = 0; i < count\_railway\_carriage; i++)

{

cout << RAILWAY\_CARRIAGE\_2 ;

}

cout << endl;

cout << " /####### ### ";

for (uint32\_t i = 0; i < count\_railway\_carriage; i++)

{

cout << RAILWAY\_CARRIAGE\_3 ;

} cout << endl;

cout << " /############-";

for (uint32\_t i = 0; i < count\_railway\_carriage; i++)

{

cout << RAILWAY\_CARRIAGE\_4 ;

} cout << endl;

cout << "/© © © ©© ";

for (uint32\_t i = 0; i < count\_railway\_carriage; i++)

{

cout << RAILWAY\_CARRIAGE\_5 ;

} cout << endl;

}

Файл route.h

#pragma once

using namespace std;

template<typename T>

class List

{

private:

template<typename T>

class Node

{

public:

Node\* pNext; //указатель на следующий элемент

T data;//данные

Node(T data = T(), Node\* pNext = nullptr)

{

this->data = data;

this->pNext = pNext;

}

};

int Size;

Node<T>\* head; //указатель на начало списка

public:

int GetSize() { return Size; }

~List();

List();

void operator +(const T data);

void pop\_front();

T& operator[](const int index);

bool operator > (List& r);

bool operator <(List& r);

};

template<typename T>

List<T>::List()

{

Size = 0;

head = nullptr;

}

template<typename T>

List<T>::~List()

{

}

template<typename T>

T& List<T>::operator[](const int index)

{

Node<T>\* current = this->head;//âðåìåííàÿ ïåðåìåííàÿÿ, ïîêàçûâàþùàÿ êîíêðåòíûé ýëåìåíò

int counter = 0;// ïåðåìåííàÿ ñ÷åò÷èê-ýëåìåíòîâ

while (current != nullptr)

{

if (counter == index)

{

return current->data;

}

current = current->pNext;

counter++;

}

}

template<typename T>

void List<T>::operator+(const T data)

{

head = new Node<T>(data, head);

Size++;

}

template<typename T>

inline bool List<T>::operator>(List& r)

{

if (this->GetSize() > r.GetSize())

{

return true;

}

else

{

return false;

}

}

template<typename T>

inline bool List<T>::operator<(List& r)

{

if (this->GetSize() < r.GetSize())

{

return true;

}

else

{

return false;

}

}

template<typename T>

inline void List<T>::pop\_front()

{

Node<T>\* temp = head;

head = head->pNext;

delete temp;

Size--;

}

template<typename T>

inline std::ostream& operator<<(std::ostream& os, List<T>& l)

{

os << l.GetSize()<<endl ;

for (int i = 0; i < l.GetSize(); i++)

os << l[i];

return os;

}

template<typename T>

inline std::ostream& Route\_show(std::ostream& os, List<T>& l)

{

os << " Route:\n Length:";

os << l.GetSize()<<endl;

for (int i = 0; i < l.GetSize(); i++)

City\_show(os, l[i]);

return os;

}

template<typename T>

inline std::istream& operator>>(std::istream& os, List<T>& l)

{

int len\_;

os >> len\_;

for (int i = 0; i < len\_; i++)

{

City \*c = new City;

os >> c->name;

os >> c->cargo\_size;

os >> c->delicery\_hours;

l + (\*c);

}

return os;

}

template<typename T>

inline std::istream& Route\_input(std::istream& os, List<T>& l)

{

int len\_;

cout << "input cities count\n";

os >> len\_;

for (int i = 0; i < len\_; i++)

{

City \*c = new City;

cout << "input city name\n";

os >> c->name;

cout << "input city cargo\_size\n";

os >> c->cargo\_size;

cout << "input city delicery\_hours\n";

os >> c->delicery\_hours;

l + (\*c);

}

return os;

}

Файл city.h

#pragma once

#include "lib.h"

using namespace std;

typedef struct City

{

string name;

double cargo\_size;

double delicery\_hours;

} City;

inline std::ostream& operator<<(std::ostream& os, City& c)

{

os << c.name << endl;

os << c.cargo\_size << endl;

os << c.delicery\_hours << endl;

return os;

}

inline std::ostream& City\_show(std::ostream& os, City& c)

{

os << " Name: ";

os << c.name << endl;

os << " Cargo size: ";

os << c.cargo\_size << endl;

os << " Delicery hours: ";

os << c.delicery\_hours << endl;

return os;

}

1. **Результаты работы программы**

Пример двух последовательных запусков программы при которых сначала создается машина с маршрутом на 4 города, и поезд с маршрутом на 0 городов, далее все это сохраняется и происходит выход из программы. При следующем запуске данные извлекаются из файла и информация о поезде удаляется. Сохраненную информацию можно наблюдать в data.txt.

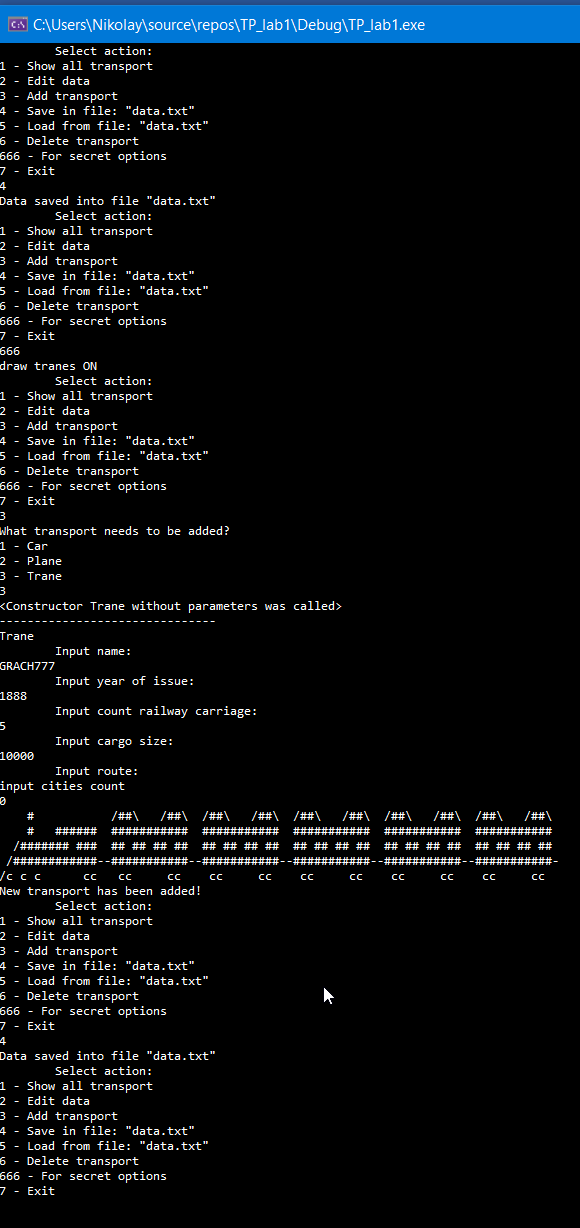
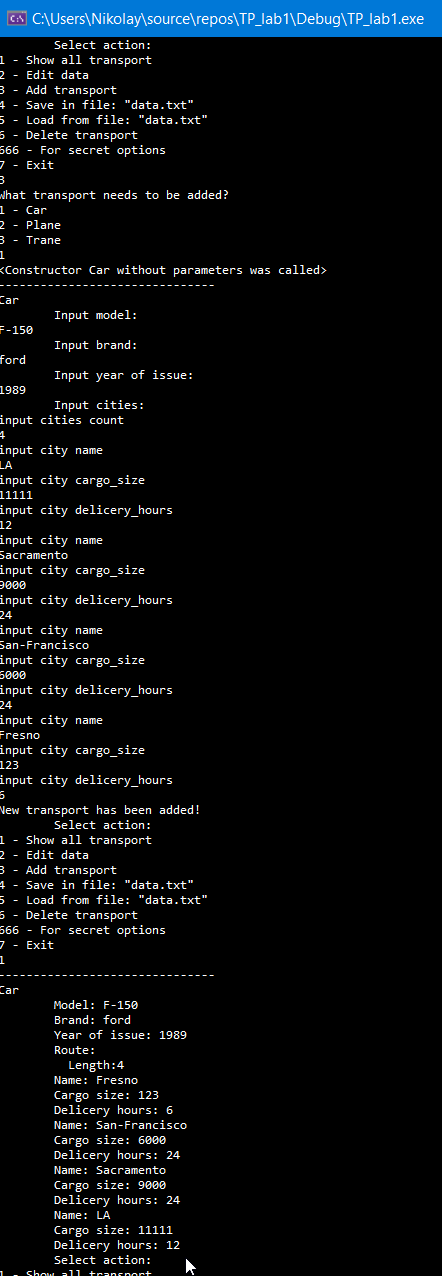


Рисунок 3. Скриншот экрана консоли

Рисунок 2. Скриншот экрана консоли

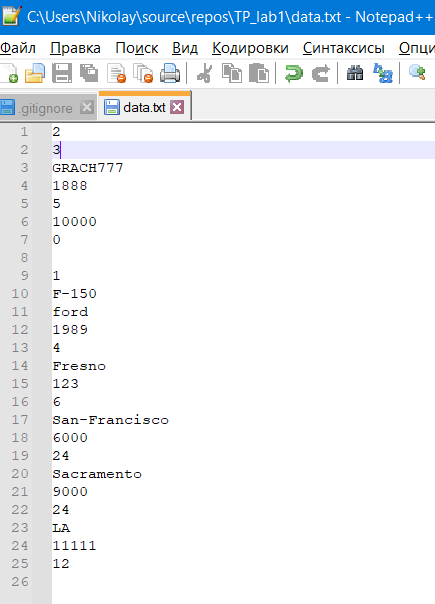
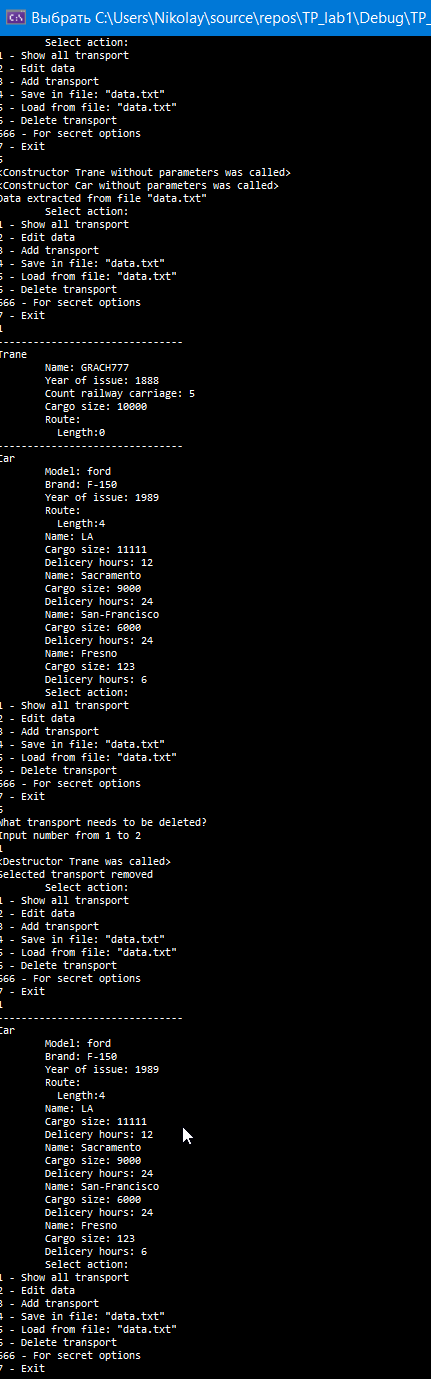


Рисунок 4. Скриншот содержимого текстового файла

Рисунок 5. Скриншот экрана консоли

1. **Выводы**

В процессе выполнения лабораторной работы мы применили навыки поученные в процессе выполнения предыдущих лабораторных работ. Изучили принципы написания программ на языке C++. Разработали программу, согласно указанным требованиям и дополнили ее самостоятельно придуманным функционалом. Программа работает корректно, что видно из представленных результатов работы.