

**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Новосибирский государственный университет экономики и управления «НИНХ»**

**(ФГБОУ ВО «НГУЭУ», НГУЭУ)**

Кафедра информационных технологий

Консольное приложение «Заказы компании перевозок»

**РАСЧЕТНО-ГРАФИЧЕСКАЯ РАБОТА**

Дисциплина: Языки программирования

Ф.И.О студента: Семенов Руслан Евгеньевич

Направление: 02.03.02 Фундаментальная информатика и информационные технологии

Направленность (профиль): Программная инженерия

Номер группы: ФИ202

Номер зачетной книжки: 220137

Номер варианта расчетно-графической работы: 7

Проверил: Ковригин Алексей Викторович, кандидат педагогических наук, доцент

Новосибирск 2024

# **Содержание**

[Содержание 2](#_Toc155563750)

[Выполнение и реализация поставленной задачи 3](#_Toc155563751)

[Глава 1. Создание 3 объектов сущности 4](#_Toc155563752)

[Глава 2.1 Меню консольного приложения 7](#_Toc155563753)

[Глава 2.2 Класс transportation\_service 10](#_Toc155563754)

[Глава 2.3 Класс record 13](#_Toc155563755)

[Глава 2.4 Класс people 14](#_Toc155563756)

[Глава 3 Список тестовых случаев для ручного тестирования программы 15](#_Toc155563757)

# 

# **Выполнение и реализация поставленной задачи**

Поставленная задача – написание консольного приложения на языке программирования С++. Программа должна уметь выполнять следующие задачи:

1. Создавать 3 объекта сущности.
2. Указывать взаимодействия объектов сущности.
3. Иметь интерфейс ввода и вывода из программы.
4. Возможность редактирования сущностей.
5. Ручной ввод сущностей.
6. Сохранение и выгрузка сущностей из структурированного текстового файла.
7. Удаление выбранных сущностей.

Консольное приложение – это приложение, разработанное для работы в текстовом пользовательском интерфейсе программы. Примером может служить командная строка некоторых операционных систем таких как Windows или Linux.

# **Глава 1. Создание 3 объектов сущности**

Первое, происходит при запуске программы – создание 3 объектов сущностей.

class transportation\_service{

private:

int con;

record\* orders;

public:

transportation\_service() {

con = 3;

setlocale(LC\_ALL, "ru");

orders = new record[con];

for (int j = 0; j < con; j++) {record newobject;newobject.createOrder();orders[j] = newobject;}

}

Создаем класс «transportation\_service». Инициализируем поле con со значением 3, устанавливаем локальную настройку для вывода на русском языке, создаем массив объектов типа record с помощью оператора new и заполняем его объектами newobject, созданными с помощью оператора new и функции createOrder.

class record {

private:

people client;

string date, adress;

string type\_transport[3] = {"Морское судно","Грузовой самолет","Газель"};

string transport;

float weight;

public:

record() {date = "01.01.1999";adress = "Новосибирск,Советская,40";transport = "Газель";weight = 100;}

record(string name, string surname, string number, string transport, float weight, string date, string adress) {

this->client = \*(new people(name, surname, number));

this->weight = weight;

this->date = date;

this->adress = adress;

this->transport = transport;

}

void createOrder() {

string name, surname, number;

cout << "Введите ваше имя: "; cin >> name;

cout << "Введите вашу фамилию: "; cin >> surname;

cout << "Введите ваш номер телефона: "; cin >> number;

client = \*(new people(name, surname, number));

cout << "Выберите желаемый транспорт доставки:" << endl;

int select = 0;

for (int i = 0; i < 3; i++) {cout << "[" << i + 1 << "] " << type\_transport[i] << endl;}

while (select < 1 || select>3) {cin >> select;}

transport = type\_transport[select - 1];

cout << "Укажите общий вес перевозки: ";

int newweight = 0;

while (newweight < 1) {cin >> newweight;}

weight = newweight;

string newdate;

cout << "Введите дату погрузки: ";

do{cin >> newdate;} while (!(check\_date(newdate)));

date = newdate;

string city, street, house;

cout << "Введите город: "; cin >> city;

cout << "Введите улицу: "; cin >> street;

cout << "Введите номер дома: "; cin >> house;

adress = city + "," + street + "," + house;

}

string getTransport() {return transport;}

float getWeight() {return weight;}

string getDate() {return date;}

string getAdress() {return adress;}

people getClient() {return client;}

string getClientInfo() {client.print();}

bool check\_year(int y) {

if (y % 400 == 0)return true;

else {

if (y % 100 == 0)return false;

else {

if (y % 4 == 0)return true;

else return false;

}

}

}

bool check\_date(string date) {

int p1 = date.find('.');

int p2 = date.find('.', p1 + 1);

if (p1 == -1 || p2 == -2) {cout << "Некорректная дата." << endl;return false;}

int d = stoi(date.substr(0, p1));

int m = stoi(date.substr(p1 + 1, p2 - p1));

int y = stoi(date.substr(p2 + 1, date.length() - p2));

if (y < 0 || m>12 || m < 1 || d < 1) {cout << "Некорректная дата." << endl;return false;}

map<string, int> days;

days["1"] = 31;

if (check\_year(y))days["2"] = 29;

else days["2"] = 28;

days["3"] = 31;

days["4"] = 30;

days["5"] = 31;

days["6"] = 30;

days["7"] = 31;

days["8"] = 31;

days["9"] = 30;

days["10"] = 31;

days["11"] = 30;

days["12"] = 31;

if (d > days[to\_string(m)]){cout << "Некорректная дата." << endl;return false;}

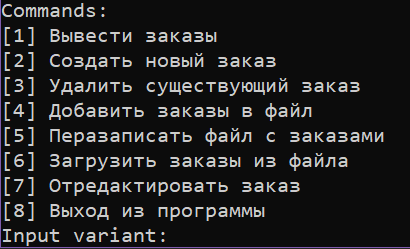
return true;

}

В этой части кода создается класс Record. В нем объявляются приватные переменные класса record: people client, string date, adress, transport, float weight. Устанавливается значения полей date, adress, transport и weight в конструкторе по умолчанию. Объявляется конструктор record, который принимает параметры: name, surname, number, transport, weight, date, adress и инициализирует переменные класса record с их значениями. Объявляется метод createOrder, который запрашивает у пользователя информацию о заказе и сохраняет ее в полях класса record. Проверяется корректность введенной даты с помощью функции check\_date. Объявляюся дополнительные переменные для удобства работы с методом createOrder: name, surname, number, transport, weight, city, street, house, adress. Объявляются функции класса record: getTransport, getWeight, getDate, getAdress, getClient, getClientInfo, которые возвращают соответствующие переменные. Функция check\_year проверяет, является ли год високосным, а функция check\_date проверяет, является ли введенная дата корректной. Для проверки используются константы дней в месяце для не високосного года. Если год високосный, то в словаре days для февраля добавляется значение 29. Затем проверяется, не превышает ли количество дней в месяце, указанное в введенной дате, количество дней в этом месяце для данного года. Если превышает, то выводится сообщение о некорректной дате и возвращается значение false. Если все проверки пройдены успешно, то возвращается значение true.

# **Глава 2.1 Меню консольного приложения**

После создание 3 объектов сущностей появится меню консольного приложения.



1 рисунок - панель меню

Код меню:

#include <iostream>

#include "orders.h"

using namespace std;

class controller {

private:

transportation\_service head\_service;

public:

controller() {}

void execute() {

int run = 1;

while (run) {

menu();

int var = 0;

while (var < 1 || var > 8) {cout << "Input variant: ";cin >> var;}

if (var == 8) {run = 0;}

if (var == 1) {head\_service.print();}

if (var == 2) {head\_service.addRecord();cout << "Заказ успешно создан!";}

if (var == 3) {head\_service.print();cout << "Введите номер заказа, которой надо удалить:\n";int n;cin >> n; head\_service.deleteRecord(n);}

if (var == 4) {head\_service.savingToFile(true);cout << "Заказы успешно сохранены!" << endl;}

if (var == 5) {head\_service.savingToFile(false);}

if (var == 6) {head\_service.loadFiles();cout << "Заказы из файла успешно загружены!" << endl;}

if (var == 7) {cout << "Выберите заказ, который следует отредактировать:" << endl;

int choose = 0;

while (choose < 1 || choose> head\_service.getCon() + 1) {

head\_service.print();

cin >> choose;

}

head\_service.editRecord(choose);

cout << "Заказ успешно отредактирован!" << endl;

}

}

}

void menu() {

cout << "Commands: " << endl;

cout << "[1] Вывести заказы" << endl;

cout << "[2] Создать новый заказ" << endl;

cout << "[3] Удалить существующий заказ" << endl;

cout << "[4] Добавить заказы в файл" << endl;

cout << "[5] Перазаписать файл с заказами" << endl;

cout << "[6] Загрузить заказы из файла" << endl;

cout << "[7] Отредактировать заказ" << endl;

cout << "[8] Выход из программы" << endl;

}

};

Данный код представляет собой контроллер для меню заказов. Он включает в себя несколько функций для работы с транспортной службой и файлами.

Класс controller имеет приватную переменную head\_service, которая является объектом класса transportation\_service. У этого класса есть публичные функции execute(), menu() и конструктор.

Функция execute() является главным циклом программы. Она использует цикл while для повторения меню, пока пользователь не введет 8 для выхода из программы.

Внутри цикла while есть вложенный цикл while, который используется для получения допустимого ввода от пользователя.

* Если пользователь вводит 1, выводится информация о текущих заказах.
* Если пользователь вводит 2, создается новый заказ и выводится сообщение об успехе.
* Если пользователь вводит 3, удаляется заказ и выводится сообщение об успехе.
* Если пользователь вводит 4, заказы добавляются в файл и выводится сообщение об успехе.
* Если пользователь вводит 5, файл с заказами перезаписывается и выводится сообщение об успехе.
* Если пользователь вводит 6, загружаются заказы из файла и выводится сообщение об успехе.
* Если пользователь вводит 7, выбирается заказ для редактирования, и после ввода номера заказа, выводится сообщение об успехе.

Функция menu() просто выводит меню заказов на экран.

# **Глава 2.2 Класс transportation\_service**

Класс transportation\_service содержит функции для работы с заказами, print(), addRecord(), deleteRecord(), savingToFile(), loadFiles() и editRecord(). Класс transportation\_service имеет функции-члены для печати информации о заказах, добавления нового заказа, удаления заказа, сохранения заказов в файл, загрузки заказов из файла и редактирования заказа.

#include <iostream>

#include <string>

#include "record.h"

#include <vector>

#include <fstream>

using namespace std;

class transportation\_service{

private:

int con;

record\* orders;

public:

transportation\_service() {

con = 3;

setlocale(LC\_ALL, "ru");

orders = new record[con];

for (int j = 0; j < con; j++) {record newobject;newobject.createOrder();orders[j] = newobject;}

}

void print() {

setlocale(LC\_ALL, "ru");

for (int i = 0; i < con; i++) {record head\_record = orders[i];cout << "[" << i + 1 << "] ";head\_record.print();}

}

void addRecord() {

record newrc;

newrc.createOrder();

record\* newor = new record[con + 1];

for (int i = 0; i < con; i++) {newor[i] = orders[i];}

newor[con] = newrc;

con++;

orders = newor;

}

void addRecordFile(record& newrc) {

record\* newor = new record[con + 1];

for (int i = 0; i < con; i++) {newor[i] = orders[i];}

newor[con] = newrc;

con++;

orders = newor;

}

void deleteRecord(int k) {

record\* newor = new record[con - 1];

for (int i = 0, j = 0; i < con; i++, j++) {if (i == (k - 1)) { i++; }if (i < con) { newor[j] = orders[i]; }}

con--;

orders = newor;

}

void savingToFile(bool app) {

ofstream out;

try {

if (app) { out.open("orders.txt", ios::app); }

else { out.open("orders.txt"); }

}

catch (const char\* error\_message) {cout << error\_message << endl;return;}

if (out.is\_open()) {

string rcInfo;

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

record cr\_rc = orders[i];

rcInfo = cr\_rc.getClient().getName() + "||" + cr\_rc.getClient().getSurname() + "||" + cr\_rc.getClient().getNumber() + "||" + cr\_rc.getTransport() + "||" + to\_string(cr\_rc.getWeight()) + "||" + cr\_rc.getDate() + "||" + cr\_rc.getAdress();

out << rcInfo << endl;

}

out.close();

}

}

void loadFiles() {

ifstream in("orders.txt");

if (in.is\_open()){

string newrc;

while (getline(in, newrc)) {

string name = newrc.substr(0, newrc.find("||"));

newrc = newrc.substr(newrc.find("||") + 2, newrc.length() - newrc.find("||"));

string surname = newrc.substr(0, newrc.find("||"));

newrc = newrc.substr(newrc.find("||") + 2, newrc.length() - newrc.find("||"));

string number = newrc.substr(0, newrc.find("||"));

newrc = newrc.substr(newrc.find("||") + 2, newrc.length() - newrc.find("||"));

string transport = newrc.substr(0, newrc.find("||"));

newrc = newrc.substr(newrc.find("||") + 2, newrc.length() - newrc.find("||"));

float weight = stof(newrc.substr(0, newrc.find("||")));

newrc = newrc.substr(newrc.find("||") + 2, newrc.length() - newrc.find("||"));

string date = newrc.substr(0, newrc.find("||"));

newrc = newrc.substr(newrc.find("||") + 2, newrc.length() - newrc.find("||"));

string adress = newrc.substr(0, newrc.length());

record\* newrc = new record(name, surname, number, transport, weight, date, adress);

addRecordFile(\*newrc);

}

}

in.close();

}

void editRecord(int con) {orders[con - 1].edit();}

int getCon() {return con;}

};

У класса есть приватные переменные con (количество заказов) и orders (массив объектов record, представляющих заказы).

Функция print() класса transportation\_service выводит информацию о текущих заказах на экран.

В начале функции устанавливается русский язык с помощью функции setlocale().

Функция addRecord() добавляет новый заказ в массив orders. Если массив orders полон, то создается новый массив newor, в который копируются все элементы из старого массива orders, а затем добавляется новый заказ. Старый массив orders заменяется новым массивом newor.

Функция deleteRecord(int k) удаляет заказ с индексом k из массива orders. Если k равен последнему индексу в массиве, то индекс не изменяется. Если k меньше последнего индекса в массиве, то индекс k заменяется на k + 1. Если k больше или равен нулю, то создается новый массив newor, в который копируются все элементы из старого массива orders, начиная с индекса 1 (не включая k). Затем старый массив orders заменяется новым массивом newor.

Функция savingToFile(bool app) сохраняет информацию о заказах в файл orders.txt. Если параметр app равен true, то файл открывается в режиме добавления (открывается для записи и позиция записи перемещается в конец файла). Если параметр app равен false, то файл открывается в режиме чтения (открывается для чтения и позиция чтения перемещается в начало файла). Если файл не может быть открыт, то выводится сообщение об ошибке и программа завершается.

Функция loadFiles() загружает информацию о заказах из файла orders.txt. Если файл не может быть открыт, то программа завершается.

Функция editRecord(int con) редактирует заказ с индексом con в массиве orders.

# **Глава 2.3 Класс record**

Класс record представляет собой объект, содержащий информацию о заказе, такую как имя клиента, фамилия клиента, номер телефона клиента, вид транспорта, вес заказа, дата заказа и адрес доставки. Класс record имеет функции createOrder(), print(), edit(), getClient(), getSurname(), getNumber(), getTransport(), getWeight(), getDate(), getAdress(). Он имеет приватные переменные client, weight, date и adress, которые хранят информацию о клиенте, весе заказа, дате заказа и адресе доставки соответственно. Также у класса есть приватный массив type\_transport, который содержит список доступных видов транспорта.

Класс record имеет конструктор, который принимает параметры name, surname, number, transport, weight, date и adress и использует их для инициализации переменных объекта.

У класса record есть метод createOrder(), который запрашивает у пользователя информацию о заказе и использует ее для создания объекта record.

Метод print() выводит информацию о заказе на экран. Он вызывает метод print() объекта client, чтобы вывести информацию о клиенте, а затем выводит информацию о заказе, включая тип транспорта, вес заказа, дату заказа и адрес доставки.

У класса record также есть метод edit(), который запрашивает у пользователя, какой пункт заказа он хотел бы отредактировать, и в зависимости от выбора пользователя выполняет соответствующую операцию редактирования.

# **Глава 2.4 Класс people**

#include <iostream>

#include <string>

using namespace std;

class people {

private:

string name, surname, number;

public:

string getName() {return name;}

string getSurname() {return surname;}

string getNumber() {return number;}

people() {name = "Руслан";surname = "Семенов";number = "89137776655";}

people(string name, string surname, string number) {this->name = name;this->surname = surname;this->number = number;}

void print() {cout << surname << " " << name << " - " << number << endl;}

};

Класс people имеет приватные переменные name, surname и number, которые хранят информацию о имени, фамилии и номере телефона человека.

Класс people имеет конструктор, который принимает параметры name, surname и number и использует их для инициализации переменных объекта.

Метод print() класса people выводит информацию о человеке на экран.

Таким образом, класс people предоставляет информацию о человеке, включая его имя, фамилию и номер телефона. Этот класс для хранения информации о клиентах и их заказах.

# **Глава 3 Список тестовых случаев для ручного тестирования программы**

|  |  |  |
| --- | --- | --- |
| № | Описание тестового случая (выполняемые действия) | Ожидаемый результат |
| 1 | Запуск приложения | Вывод сообщений запрашивающих информацию о заказах. Будет повторятся 3 раза, для создания 3 объектов сущностей – заказов. После заполнения всех пунктов будет выведено меню с командами. |
| 2 | Вывод заказов | Выводит все имеющиеся и формирующиеся на данный момент заказы. Следующим сообщением выводит меню. |
| 3 | Создать новый заказ | Выводит сообщения необходимые для создания заказа и заполнения информации. После завершения выводит сообщение об успешном создании заказа, а так же выводит меню. |
| 4 | Удалить существующий заказ | Выводит все имеющиеся заказы, а так же сообщение с просьбой выбрать соответствующий заказ под номером. После чего выводится меню. |
| 5 | Добавить заказы в файл | Открывает или создает файл «orders.txt», добавляет в файл заказы. Выводит сообщение об успешной загрузке товаров и меню. |
| 6 | Перезаписать файл с заказами | Открывает файл с заказами «orders.txt», перепроверяет и обновляет информацию и них. Выводит меню. |
| 7 | Загрузка заказов из файла | Открывает файл с заказами «orders.txt», и загружает заказы из файла к существующим заказам для редактирования. Выводит сообщение об успешной загрузке, а так же выводится меню. |
| 8 | Редактирование заказов | Выводит все существующие заказы, и просит выбрать номер заказа который следует отредактировать. После выводится сообщение о том что именно необходимо отредактировать. Запрашиваются новые данные. Выводится сообщение об успешном редактирование заказа и меню. |
| 9 | Выход из программы | Программа прекращает работать, диалоговое окно закрывается. |