**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

Курсовая РАБОТА

**по дисциплине «Программирование»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 1302 |  | Гордиенко М. Ю |
| Преподаватель |  | Калмычков В.А |

Санкт-Петербург

2022

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент: Гордиенко М. Ю | | |
| Группа 1302 | | |
| Исходные данные: Список команд, связанных с добавлением клиентов, машин, составлением заказов.  Необходимо: Написать программу, моделирующую работу сервиса аренды машин. | | |
| Содержание пояснительной записки: цель курсовой работы, исходная постановка, способы реализации ввода-вывода, написанные функции, разбиение программы на файлы, алгоритмы и структуры данных, текст программы, тестирование программы, вывод. | | |
| Дата выдачи задания: | | |
| Дата сдачи реферата: | | |
| Дата защиты реферата: | | |
| Студент |  | Гордиенко М. Ю |
| Преподаватель |  | Калмычков В.А |

**Аннотация**

В работе представлена программа, решающая поставленную задачу по моделированию сервиса аренды автомобилей с подготовленным функционалом. На вход подается файл, со списком команд. Программой ведётся добавление клиентов, автомобилей, проверка возможности создать заказ, создание заказа, подсчет стоимости.

**Summary**

The paper presents a program that solves the task of modeling a car rental service with prepared functionality. A file with a list of commands is submitted to the input. The program is adding customers, cars, checking the ability to create an order, creating an order, calculating the cost.

Оглавление

[Цель курсовой работы: 5](#_Toc105509868)

[Исходная постановка: 5](#_Toc105509869)

[Способы реализации ввода-вывода 5](#_Toc105509870)

[Написанные функции: 5](#_Toc105509871)

[Разбиение программы на файлы 8](#_Toc105509872)

[Алгоритмы и структуры данных 9](#_Toc105509873)

[Текст программы 9](#_Toc105509874)

[Тестирование программы 16](#_Toc105509875)

[Вывод 18](#_Toc105509876)

# **Цель курсовой работы:**

Продемонстрировать умение хранить и обрабатывать списки и умение моделировать работу сервиса аренды.

# Исходная постановка:

Изображение выглядит как текст

Автоматически созданное описание

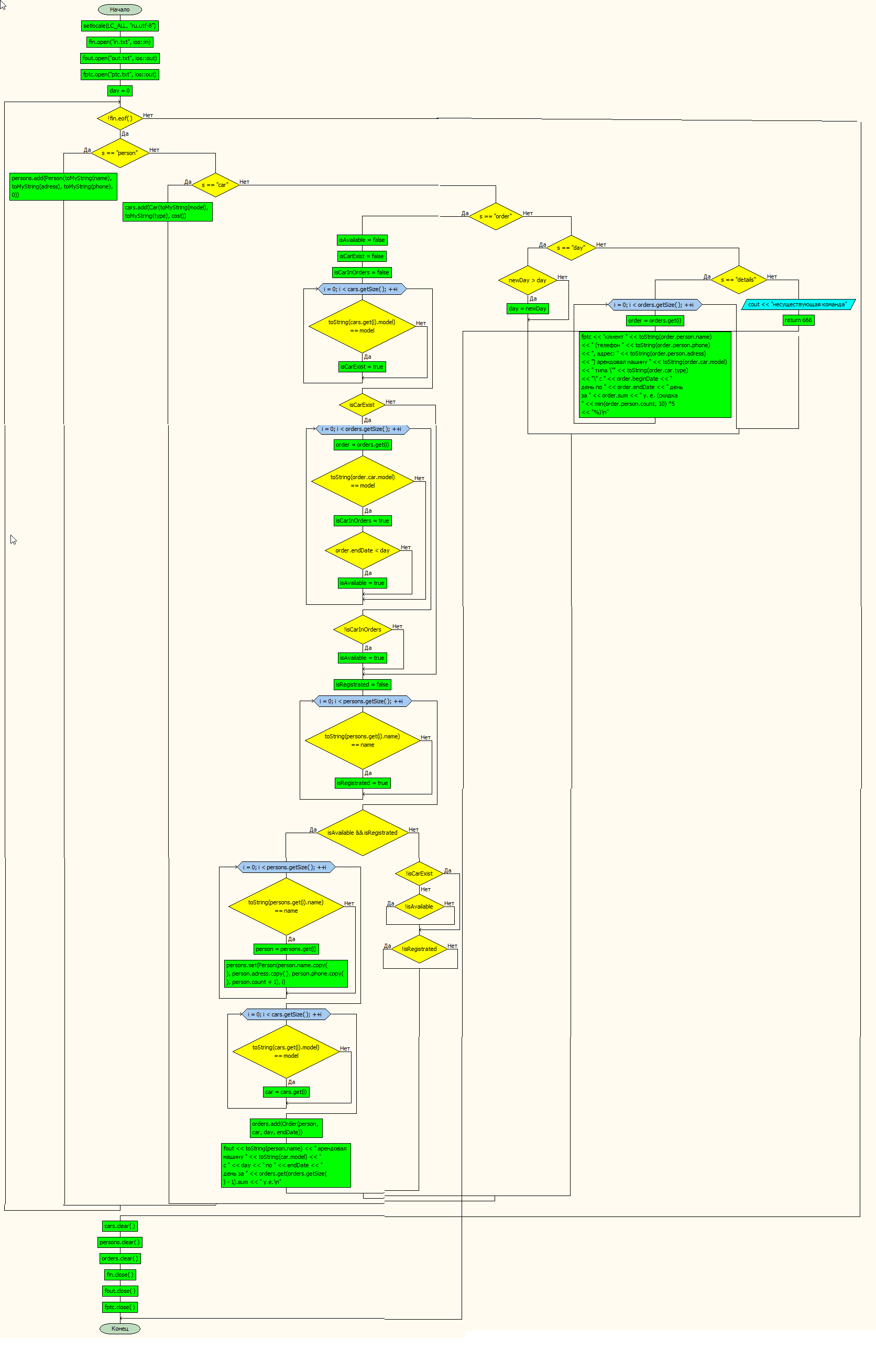
Способы реализации ввода-вывода:

|  |  |  |
| --- | --- | --- |
|  | iostream | fstream |
| Ввод | cin | ios::in |
| Вывод | cout | ios::out |

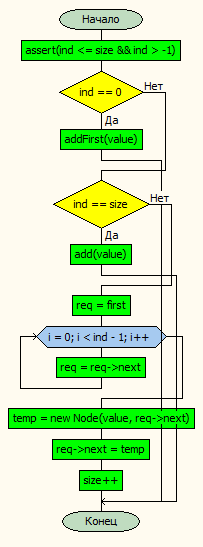
# **Написанные функции:**

|  |  |
| --- | --- |
| toMyString | Сделать список блоков из строки |
| toString | Сделать строку из списка блоков |
| getSize | Длина списка |
| add | Добавить элемент |
| get | Вернуть элемент по индексу |
| set | Заменить элемент по индексу |
| remove | Удалить элемент по индексу |
| copy | Скопировать список |
| clear | Очистить список |
| toArray | Вернуть массив из элементов списка |

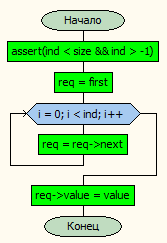
**main:**

****

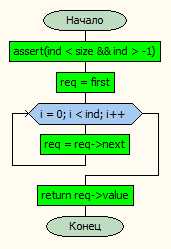
**add:**

****

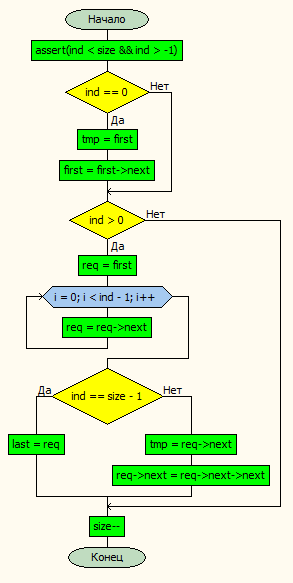
**get:**

****

**set:**

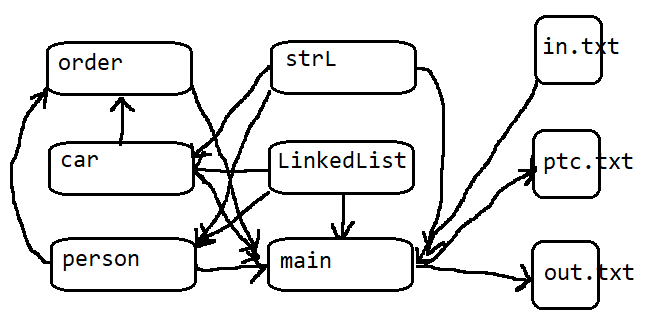
****

**remove:**

****

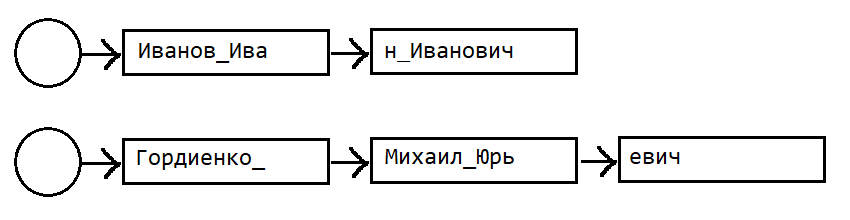
# **Разбиение программы на файлы**

|  |  |
| --- | --- |
| main.cpp | Основа программы |
| StrL.cpp | Структура слова, из которых состоит список |
| StrL.h | Интерфейс структуры слова |
| LinkedList.h | Интерфейс списка вместе с реализацией |
| LinkedList.cpp | Остается пустым, так как шаблонные классы реализуются в заголовочном файле |
| Order.h | Заголовок структуры заказа |
| Order.cpp | Структура заказа |
| Car.h | Заголовок структуры машины |
| Car.cpp | Структура машины |
| Person.h | Заголовок структуры клиента |
| Person.cpp | Структура клиента |



# Алгоритмы и структуры данных

**Список блоков символов myString:**



# Текст программы

**Main**

#include <iostream>  
#include "LinkedList.h"  
#include <string>  
#include <fstream>  
#include "Order.h"  
#include "Person.h"  
#include "Car.h"  
  
using namespace std;  
  
LinkedList<StrL> toMyString(string s) {  
 const int N = 10;  
 LinkedList<StrL> a;  
 for (int i = 0; i < (s.length() + N - 1) / N; i++) {  
 a.add(s.substr(i \* N, N));  
 }  
 return a;  
}  
  
string toString(LinkedList<StrL> list) {  
 string s = "";  
 for (int i = 0; i < list.getSize(); ++i) {  
 s += list.get(i).arr;  
 }  
 return s;  
}  
  
int main() {  
 setlocale(LC\_ALL, "ru.utf-8");  
 fstream fin;  
 fin.open("in.txt", ios::in);  
 fstream fout;  
 fout.open("out.txt", ios::out);  
 fstream fptc;  
 fptc.open("ptc.txt", ios::out);  
 LinkedList<Order> orders;  
 LinkedList<Person> persons;  
 LinkedList<Car> cars;  
 string s;  
 int day = 0;  
 while (!fin.eof()) {  
 fin >> s;  
 if (s == "person") {  
 string name;  
 string adress;  
 string phone;  
 fin >> name >> adress >> phone;  
 persons.add(Person(toMyString(name), toMyString(adress), toMyString(phone), 0));  
 fout << "клиент " << name << " был зарегистрирован\n";  
 } else if (s == "car") {  
 string model;  
 string type;  
 int cost;  
 fin >> model >> type >> cost;  
 cars.add(Car(toMyString(model), toMyString(type), cost));  
 fout << "машина " << model << " была добавлена\n";  
 } else if (s == "order") {  
 string name;  
 string model;  
 int endDate;  
 fin >> name >> model >> endDate;  
 bool isAvailable = false;  
 bool isCarExist = false;  
 bool isCarInOrders = false;  
 //наличие машины  
 for (int i = 0; i < cars.getSize(); ++i) {  
 if (toString(cars.get(i).model) == model) {  
 isCarExist = true;  
 }  
 }  
 //доступность  
 if (isCarExist) {  
 for (int i = 0; i < orders.getSize(); ++i) {  
 Order order = orders.get(i);  
 if (toString(order.car.model) == model) {  
 isCarInOrders = true;  
 if (order.endDate < day) {  
 isAvailable = true;  
 }  
 }  
 }  
 if (!isCarInOrders) {  
 isAvailable = true;  
 }  
 }  
 bool isRegistrated = false;  
 //наличие человека  
 for (int i = 0; i < persons.getSize(); ++i) {  
 if (toString(persons.get(i).name) == name) {  
 isRegistrated = true;  
 }  
 }  
 if (isAvailable && isRegistrated) {  
 Person person;  
 Car car;  
 for (int i = 0; i < persons.getSize(); ++i) {  
 if (toString(persons.get(i).name) == name) {  
 person = persons.get(i);  
 persons.set(  
 Person(person.name.copy(), person.adress.copy(), person.phone.copy(), person.count + 1),  
 i);  
 }  
 }  
 for (int i = 0; i < cars.getSize(); ++i) {  
 if (toString(cars.get(i).model) == model) {  
 car = cars.get(i);  
 }  
 }  
 orders.add(Order(person, car, day, endDate));  
 fout << toString(person.name) << " арендовал машину " << toString(car.model) << " с " << day << " по "  
 << endDate  
 << " день за " << orders.get(orders.getSize() - 1).sum << " у.е.\n";  
 } else {  
 if (!isCarExist) {  
 fout << "машина " << model << " на " << day << " день отсутствует. ";  
 } else if (!isAvailable) {  
 fout << "машина " << model << " на " << day << " день занята. ";  
 }  
 if (!isRegistrated) {  
 fout << "человек " << name << " не зарегистрирован.";  
 }  
 fout << "\n";  
 }  
 } else if (s == "day") {  
 int newDay;  
 fin >> newDay;  
 if (newDay > day) {  
 day = newDay;  
 }  
 } else if (s == "details") {  
 for (int i = 0; i < orders.getSize(); ++i) {  
 Order order = orders.get(i);  
 fptc << "клиент " << toString(order.person.name) << " (телефон " << toString(order.person.phone)  
 << ", адрес: "  
 << toString(order.person.adress) << ") арендовал машину " << toString(order.car.model)  
 << " типа \"" << toString(order.car.type) <<  
 "\" с " << order.beginDate << " день по " << order.endDate << " день за " << order.sum  
 << " у. е. (скидка " << min(order.person.count, 10) \* 5 << "%)\n";  
 }  
 fptc << '\n';  
 } else {  
 cout << "несуществующая команда";  
 return 666;  
 }  
 }  
 cars.clear();  
 persons.clear();  
 orders.clear();  
 fin.close();  
 fout.close();  
 fptc.close();  
}

**LinkedList.h**

#pragma once  
  
#include <iostream>  
#include <cassert>  
  
using namespace std;  
  
template<typename T>  
class LinkedList {  
private:  
 struct Node {  
 T value;  
 Node \*next;  
  
 Node(T value, Node \*next) {  
 this->value = value;  
 this->next = next;  
 };  
 };  
  
 int size = 0;  
 Node \*first;  
 Node \*last;  
public:  
 //~LinkedList();  
  
 int getSize();  
  
 void addFirst(T value);  
  
 void add(T value);  
  
 void add(T value, int ind);  
  
 void remove(int ind);  
  
 void set(T value, int ind);  
  
 T get(int ind);  
  
 LinkedList<T> copy();  
  
 void clear();  
  
 void print();  
  
 T \*toArray();  
};  
  
template<typename T>  
int LinkedList<T>::getSize() {  
 return size;  
}  
  
template<typename T>  
void LinkedList<T>::addFirst(T value) {  
 if (size > 0) {  
 first = new Node(value, first);  
 } else {  
 first = new Node(value, nullptr);  
 last = first;  
 }  
 size++;  
}  
  
template<typename T>  
void LinkedList<T>::add(T value) {  
 if (size > 0) {  
 last->next = new Node(value, nullptr);  
 last = last->next;  
 } else {  
 first = new Node(value, nullptr);  
 last = first;  
 }  
 size++;  
}  
  
template<typename T>  
void LinkedList<T>::add(T value, int ind) {  
 assert(ind <= size && ind > -1);  
 if (ind == 0) {  
 addFirst(value);  
 return;  
 }  
 if (ind == size) {  
 add(value);  
 return;  
 }  
 Node \*req = first;  
 for (int i = 0; i < ind - 1; i++) {  
 req = req->next;  
 }  
 Node \*temp = new Node(value, req->next);  
 req->next = temp;  
 size++;  
  
}  
  
template<typename T>  
void LinkedList<T>::set(T value, int ind) {  
 assert(ind < size && ind > -1);  
 Node \*req = first;  
 for (int i = 0; i < ind; i++) {  
 req = req->next;  
 }  
 req->value = value;  
}  
  
template<typename T>  
T LinkedList<T>::get(int ind) {  
 assert(ind < size && ind > -1);  
 Node \*req = first;  
 for (int i = 0; i < ind; i++) {  
 req = req->next;  
 }  
 return req->value;  
}  
  
template<typename T>  
LinkedList<T> LinkedList<T>::copy() {  
 LinkedList<T> b;  
 Node \*req = first;  
 for (int i = 0; i < size; i++) {  
 b.add(req->value);  
 req = req->next;  
 }  
 return b;  
}  
  
template<typename T>  
void LinkedList<T>::clear() {  
 while (size > 0) {  
 Node \*second = first->next;  
 delete first;  
 first = second;  
 size--;  
 }  
}  
  
template<typename T>  
void LinkedList<T>::print() {  
 Node \*req = first;  
 for (int i = 0; i < size; i++) {  
 cout << (req->value);  
 req = req->next;  
 }  
}  
  
template<typename T>  
T \*LinkedList<T>::toArray() {  
 T \*a = new T[getSize()];  
 Node \*req = first;  
 for (int i = 0; i < size; i++) {  
 a[i] = (req->value);  
 req = req->next;  
 }  
 return a;  
}  
  
/\*  
template<typename T>  
LinkedList<T>::~LinkedList() {  
 clear();  
}  
\*/  
template<typename T>  
void LinkedList<T>::remove(int ind) {  
 //cout << ind << "\n";  
 assert(ind < size && ind > -1);  
 if (ind == 0) {  
 Node \*tmp = first;  
 first = first->next;  
 delete tmp;  
 }  
 if (ind > 0) {  
 Node \*req = first;  
 for (int i = 0; i < ind - 1; i++) {  
 req = req->next;  
 }  
 if (ind == size - 1) {  
 last = req;  
 delete req->next;  
 } else {  
 Node \*tmp = req->next;  
 req->next = req->next->next;  
 delete tmp;  
 }  
 }  
 size--;  
}

**Order.h**

#pragma once  
  
#include <string>  
#include "Person.h"  
#include "Car.h"  
  
using namespace std;  
  
struct Order {  
 Person person;  
 Car car;  
 int beginDate;  
 int endDate;  
 int sum;  
  
 Order(Person &person, Car &car, int beginDate, int endDate);  
  
 Order() {}  
};

**Order.cpp**

#include "Order.h"  
  
Order::Order(Person &person, Car &car, int beginDate, int endDate) : person(person), car(car), beginDate(beginDate),  
 endDate(endDate) {  
 sum = (endDate - beginDate + 1) \* car.cost \* ((1.0 - min(person.count, 10) \* 0.05 ));  
}

**Car.h**

#pragma once  
  
#include <string>  
#include "LinkedList.h"  
#include "StrL.h"  
  
using namespace std;  
  
struct Car {  
 LinkedList<StrL> model;  
 LinkedList<StrL> type;  
 int cost;  
  
 Car(LinkedList<StrL> model, LinkedList<StrL> type, int cost);  
  
 Car() {}  
};

**Car.cpp**

#include "Car.h"  
  
using namespace std;  
  
Car::Car(LinkedList<StrL> model, LinkedList<StrL> type, int cost) : model(model), type(type), cost(cost) {}

**Person.h**

#pragma once  
#include <string>  
#include "LinkedList.h"  
#include "StrL.h"  
using namespace std;  
  
struct Person {  
 LinkedList<StrL> name;  
 LinkedList<StrL> adress;  
 LinkedList<StrL> phone;  
 int count;  
  
 Person(LinkedList<StrL> name, LinkedList<StrL> adress, LinkedList<StrL> phone, int count);  
  
 Person(){}  
};

**Person.cpp**

#include "Person.h"  
  
Person::Person(LinkedList<StrL> name, LinkedList<StrL> adress, LinkedList<StrL> phone, int count) : name(name), adress(adress),  
 phone(phone), count(count) {}

**StrL.h**

#pragma once  
  
#include <string>  
#include <iostream>  
  
using namespace std;  
  
struct StrL {  
 char \*arr;  
 int len;  
  
 StrL(string s);  
  
 StrL();  
};

**StrL.cpp**

#include "StrL.h"  
  
using namespace std;  
  
StrL::StrL(string s) {  
 arr = new char[s.length() + 1];  
 strcpy(arr, s.c\_str());  
 len = s.length();  
}  
  
StrL::StrL() {  
  
}

# Тестирование программы

in.txt

person Вася Пушкинская\_22 123456  
person Петя Ленина\_5 178966  
person Саша Победы\_14 986745  
car Калина легковой 500  
car Приора легковой 700  
car Камаз грузовой 2000  
day 10  
order Вася Нива 15  
order Миша Приора 50  
order Вася Калина 15  
day 11  
order Саша Калина 13  
day 20  
order Вася Калина 21  
order Саша Приора 21  
day 30  
order Вася Приора 32  
order Саша Калина 32  
person Миша Кузнецовская\_9к3 3543543543  
order Миша Приора 50  
order Вася Нива 55  
car Нива легковой 900  
order Вася Нива 55  
details

out.txt

клиент Вася был зарегистрирован  
клиент Петя был зарегистрирован  
клиент Саша был зарегистрирован  
машина Калина была добавлена  
машина Приора была добавлена  
машина Камаз была добавлена  
машина Нива на 10 день отсутствует.   
человек Миша не зарегистрирован.  
Вася арендовал машину Калина с 10 по 15 день за 3000 у.е.  
машина Калина на 11 день занята.   
Вася арендовал машину Калина с 20 по 21 день за 950 у.е.  
Саша арендовал машину Приора с 20 по 21 день за 1400 у.е.  
Вася арендовал машину Приора с 30 по 32 день за 1890 у.е.  
Саша арендовал машину Калина с 30 по 32 день за 1425 у.е.  
клиент Миша был зарегистрирован  
Миша арендовал машину Приора с 30 по 50 день за 14700 у.е.  
машина Нива на 30 день отсутствует.   
машина Нива была добавлена  
Вася арендовал машину Нива с 30 по 55 день за 19890 у.е.

protocol.txt

клиент Вася (телефон 123456, адрес: Пушкинская\_22) арендовал машину Калина типа "легковой" с 10 день по 15 день за 3000 у. е. (скидка 0%)  
клиент Вася (телефон 123456, адрес: Пушкинская\_22) арендовал машину Калина типа "легковой" с 20 день по 21 день за 950 у. е. (скидка 5%)  
клиент Саша (телефон 986745, адрес: Победы\_14) арендовал машину Приора типа "легковой" с 20 день по 21 день за 1400 у. е. (скидка 0%)  
клиент Вася (телефон 123456, адрес: Пушкинская\_22) арендовал машину Приора типа "легковой" с 30 день по 32 день за 1890 у. е. (скидка 10%)  
клиент Саша (телефон 986745, адрес: Победы\_14) арендовал машину Калина типа "легковой" с 30 день по 32 день за 1425 у. е. (скидка 5%)  
клиент Миша (телефон 3543543543, адрес: Кузнецовская\_9к3) арендовал машину Приора типа "легковой" с 30 день по 50 день за 14700 у. е. (скидка 0%)  
клиент Вася (телефон 123456, адрес: Пушкинская\_22) арендовал машину Нива типа "легковой" с 30 день по 55 день за 19890 у. е. (скидка 15%)

# Вывод

Цель работы выполнена, программа написана с достаточно удобным интерфейсом. В ходе написания программы были закреплены практические навыки программирования, обобщены знания синтаксиса языка, и продемонстрировано усвоение материала за второй семестр. Поставленная цель полностью соответствует полученному результату.