**Содержание**

[ВВЕДЕНИЕ 5](#_Toc90066765)

[1 ОБЗОР ЛИТЕРАТУРЫ 6](#_Toc90066766)

[1.1 Обзор методов и алгоритмов решения поставленной задачи 6](#_Toc90066768)

[1.2 Постановка задачи 7](#_Toc90066769)

[2 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 8](#_Toc90066770)

[2.1 Структура входных и выходных данных 8](#_Toc90066771)

[2.2 Разработанная диаграмма классов 8](#_Toc90066772)

[2.3 Описание классов 9](#_Toc90066773)

[3 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 2](#_Toc90066780)3

[3.1 Разработка схем алгоритмов 2](#_Toc90066781)3

[3.2 Разработка алгоритмов 2](#_Toc90066782)3

[4 РЕЗУЛЬТАТЫ РАБОТЫ 2](#_Toc90066785)4

[РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 2](#_Toc90066786)6

[ЗАКЛЮЧЕНИЕ 2](#_Toc90066786)7

[ЛИТЕРАТУРА 2](#_Toc90066787)8

[ПРИЛОЖЕНИЕ А 2](#_Toc90066788)9

[ПРИЛОЖЕНИЕ Б 3](#_Toc90066789)0

[ПРИЛОЖЕНИЕ В 3](#_Toc90066790)1

[ПРИЛОЖЕНИЕ Г 3](#_Toc90066791)2

[ПРИЛОЖЕНИЕ Д 6](#_Toc90066792)1

**ВВЕДЕНИЕ**

На сегодняшний день автобусные парки очень важны, как источник общественного транспорта. Но работа по учёту автобусов на многих местах ведётся без автоматизированного процесса. Множество навыков, полученных в ходе изучения курса предмета “Конструирование Программ и Языки Программирования” вполне способны помочь справиться с подобного рода задачей.

Объектно-ориентированное программирование представляет собой технологию программирования, которая базируется на классификации и абстракции объектов. Одним из наиболее популярных средств объектно-ориентированного программирования, позволяющим разрабатывать программы, эффективные по объёму кода и скорости выполнения является С++.

С++ — компилируемый, статически типизируемый язык общего назначения. Он поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. С++ сочетает в себе как возможности низкоуровневых языков программирования, так и возможности высокоуровневых.

Основными концепциями объектно-ориентированного программирования являются инкапсуляция, полиморфизм и наследование. Язык C++ предоставляет исчерпывающие возможности для реализаций этих концепций. В результате использования инкапсуляции, программа, написанные на C++, обладает повышенной защищённостью объектов от влияния на них кода других частей этой же программы. Наследование предоставляет важную возможность повторного использования кода, что может значительно уменьшить количество кода, однако требует предварительного проектирования архитектуры. Полиморфизм позволяет программе быть более гибкой. Такая система удобна в тестировании и позволяет легко заменять и модифицировать свои компоненты.

Универсальность и гибкость языка позволяют использовать его в различных целях. Область применения данного языка включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (игр). Исходя из этого, можно считать, что данный язык достаточно удобен для написания выбранной курсового проекта.

**1 ОБЗОР ЛИТЕРАТУРЫ**

* 1. **Обзор методов и алгоритмов решения поставленной задачи**

Механизм шаблонов встроен в компилятор C++, чтобы дать возможность программистам делать свой код короче за счет обобщенного программирования. Естественно, существуют и стандартные библиотеки, реализующие этот механизм. STL является самой эффективной библиотекой C++ на сегодняшний день.

В курсовом проекте был реализован свой контейнер – list. list — коллекция, хранящая элементы в виде двунаправленного связанного списка. Также для него реализован класс итераторов и класс алгоритмов.

Для контейнера определены следующие операции:

- push – добавление нового элемента по индексу;

- pushHead – добавление нового элемента в начало;

- pushTail – добавление нового элемента в конец;

- pop – удаление элемента по индексу;

- popHead – удаление элемента из начала;

- popTail – удаление элемента из конца;

- output – просмотр всех элементов списка;

- size – размер списка;

В классе алгоритм представлены следующие алгоритмы, которые помогают решать поставленную задачу:

- registration – алгоритм, позволяющий добавить нового пользователя в систему, заполняя поля логин и пароль, а так же проверяет наличие этого пользователя в системе, тем самым отказывает в регистрации уже имеющегося пользователя.

- authorization – алгоритм, позволяющий войти в профиль зарегистрированному пользователю и получить дополнительные возможности.

- giveAdminRole – алгоритм, позволяющий присвоить права администратора пользователю, дав ему наибольшее количество представленных действий, в том числе редактировать, добавлять, удалять и отменять последние действия с автомобилями или пользователями, смотреть и удалять имеющиеся путёвки, а так же назначать других пользователей администраторами.

- sort – сортировка контейнера по полю, выбранному пользователем.

- update – алгоритм редактирования любого объекта по заданным параметрам, введённым пользователем.

* 1. **Постановка задачи**

Программа должна иметь удобный пользовательский интерфейс с необходимыми пунктами меню. Информация должна храниться в нескольких файлах, связанных определенным образом. Например, информация, содержащаяся в файлах: статус автобуса, количество мест, наличие разных особенностей. Выдаваемая информация должна быть отсортирована по разным признакам. При реализации операции редактирования, добавления, удаления информации необходимо предусмотреть операцию отмены последних действий. Разработать иерархию классов с использованием наследования. Разработать и использовать в программе классы контейнеров, итераторов и алгоритмов (свои и STL). Производить обработку исключительных ситуаций.

Для реализации программы используется объектно-ориентированный язык программирования C++, среда разработки Visual Studio 2019. Ограничения на использование других операционных систем нет. Приложение написано на ОС Windows 10.

Для работы с информацией необходимо разработать класс-контейнер «Список», который будет в себя включать добавление, удаление, изменение, доступ, сортировку и вывод. Вся информация будет хранится в файлах, поэтому необходимо разработать класс File для чтения и записи данных. Для проверки корректного ввода информации необходимо разработать класс InputError, который будет ловить и обрабатывать исключения. Так же необходимо сделать класс исключения ListException для обработки исключений в списке.

**2 СТРУКТУРА ВХОДНЫХ И ВЫХОДНЫХ ДАННЫХ**

## 2.1 Структура входных и выходных данных

В программе используется 6 текстовых файлов:

- Файл users.txt. Хранит данные о пользователях системы (логин, пароль, наличие прав администратора):

|  |
| --- |
| user|G1\_c|0  % |

- Файл trip.txt. Хранит данные о поездках (пункт назначение, расстояние, дата):

|  |
| --- |
| Brest  123 2 24  % |

- Файл microbus.txt. Хранит данные о микробусах:

|  |
| --- |
| 32 744545 12 56  % |

- Файл machine.txt. Хранит данные о машинах:

|  |
| --- |
| Заносит на поворотах| 12 425633 120 22  % |

- Файл bus.txt. Хранит данные об автобусах:

|  |
| --- |
| Заносит на поворотах| 12 425633 120 22 3 1  % |

- Файл electrobus.txt. Хранит данные об электробусах:

|  |
| --- |
| Заносит на поворотах| 12 425633 120 22 3 1 1 0 1  % |

Также в ходе работы программы для хранения промежуточных данных используются STL-контейнеры и контейнер List (см. описание классов), написанный вручную. При изменении данных в программе, происходит изменение в файле после закрытия программы.

## 2.2 Разработка диаграммы классов

Разработанная диаграмма классов представлена в приложении А.

## 2.3 Описание классов

Описание классов представлено в виде таблиц 2.3.1 - 2.3.15.

*Таблица 2.3.1 – Класс Ticket (родительский класс билета)*

|  |  |  |  |
| --- | --- | --- | --- |
| Protected поля класса | | | |
| Имя | Тип | | Описание |
| Sum | int | | Сумма для оплаты |
| id | int | | Номер |
| Public методы класса | | | |
| Имя | Принимаемые параметры | Возвращаемое значение | Описание |
| setSum | int Sum | void | Устанавливает значение Sum |
| setId | int Id | void | Устанавливает значение Id |
| getSum | void | int | Возвращает значение Sum |
| getId | void | int | Возвращает значение Id |
| operator >> | istream& is, Ticket& sub | istream | Инициализирует объект класса Ticket |
| operator << | ostream& os, const Ticket& sub | ostream | Выводит на экран объект класса Ticket |

*Таблица 2.3.2 – Класс Machine (предназначен для работы с машинами)*

|  |  |  |  |
| --- | --- | --- | --- |
| Protected поля класса | | | |
| Имя | Тип | | Описание |
| day | int | | Номер дня поездки |
| numberOfSeats | int | | Количество мест |
| features | Char[50] | | Особенности |
| Public методы класса | | | |
| Имя | Принимаемые параметры | Возвращаемое значение | Описание |
| setDay | int Day | void | Устанавливает значение Day |
| setNumberOfSeats | int NumberOfSeats | void | Устанавливает значение NumberOfSeats |
| getDay | void | int | Возвращает значение Day |
| getNumberOfSeats | void | int | Возвращает значение NumberOfSeats |
| getFeatures | void | Const char\* | Возвращает значение features |
| operator >> | istream& is, Machine& obj | istream | Инициализирует объект класса Machine |
| operator << | ostream& os, const Machine& obj | ostream | Выводит на экран объект класса Machine |

*Таблица 2.3.3 – Класс Bus (предназначен для работы с автобусами)*

|  |  |  |  |
| --- | --- | --- | --- |
| Protected поля класса | | | |
| Имя | Тип | | Описание |
| status | int | | Статус состояния |
| type | bool | | Тип |
| Public методы класса | | | |
| Имя | Принимаемые параметры | Возвращаемое значение | Описание |
| setStatus | int status | void | Устанавливает значение status |
| getStatus | void | int | Возвращает значение Status |
| isType | void | bool | Возвращает значение Type |
| operator >> | istream& is,  Bus & obj | istream | Инициализирует объект класса Bus |
| operator << | ostream& os,  const Bus & obj | ostream | Выводит на экран объект класса Bus |

*Таблица 2.3.4 – Класс Microbus (для работы с микроавтобусами)*

|  |  |  |
| --- | --- | --- |
| Private поля класса | | |
| Имя | Тип | Описание |
| numberOfCompletedTrips | int | Количество выполненных поездок |
| brand | int | Номер бренда |

*Продолжение таблицы 2.3.4*

|  |  |  |  |
| --- | --- | --- | --- |
| Public методы класса | | | |
| Имя | Принимаемые параметры | Возвращаемое значение | Описание |
| getNumberOfCompletedTrips | void | int | Возвращает значение NumberOfCompletedTrips |
| getBrand | void | int | Возвращает значени brand |
| setNumberOfCompletedTrips | int | void | Устанавливает значение NumberOfCompletedTrips |
| setBrand | int | void | Устанавливает значение brand |
| operator >> | istream& is,  Microbus& obj | istream | Инициализирует объект класса Microbus |
| operator << | ostream& os,  const Microbus& obj | ostream | Выводит на экран объект класса Microbus |

*Таблица 2.3.5 – Класс Electrobus (для работы с электробусами)*

|  |  |  |
| --- | --- | --- |
| Private поля класса | | |
| Имя | Тип | Описание |
| harmonic | bool | Наличие гармошки |
| ecological | bool | Экологичность |
| wirelessCharger | bool | Беспроводная зарядка |

*Продолжение таблицы 2.3.5*

|  |  |  |  |
| --- | --- | --- | --- |
| Public методы класса | | | |
| Имя | Принимаемые параметры | Возвращаемое значение | Описание |
| isHarmonic | void | bool | Возвращает значение Harmonic |
| isWirelessCharger | void | bool | Возвращает значение wirelessCharger |
| isEcological | void | bool | Возвращает значение Ecological |
| operator >> | istream& is,  Electrobus& obj | istream | Инициализирует объект класса Electrobus |
| operator << | ostream& os,  const Electrobus& obj | ostream | Выводит на экран объект класса Electrobus |

*Таблица 2.3.6 – Класс User (предназначен для работы с пользователями)*

|  |  |  |
| --- | --- | --- |
| Private поля класса | | |
| Имя | Тип | Описание |
| username | string | Имя пользователя |
| password | string | Пароль |
| admin | bool | Права администратора |

*Продолжение таблицы 2.3.6*

|  |  |  |  |
| --- | --- | --- | --- |
| Public методы класса | | | |
| Имя | Принимаемые параметры | Возвращаемое значение | Описание |
| getUsername | void | string | Возвращает значение username |
| getPassword | void | string | Возвращает значение password |
| isAdmin | void | bool | Возвращает значение admin |
| setUsername | string | void | Устанавливает значение username |
| setPassword | string | void | Устанавливает значение password |
| setAdmin | bool | void | Устанавливает значение admin |
| operator >> | istream& is,  User& obj | istream | Инициализирует объект класса User |
| operator << | ostream& os,  const User& obj | ostream | Выводит на экран объект класса User |
| operator= | User& | User& | Перегрузка оператора присваивания |

*Таблица 2.3.7 – Базовый класс MyException (для обработки базовых исключений)*

|  |  |  |  |
| --- | --- | --- | --- |
| Protected поля класса | | | |
| Имя | Тип | | Описание |
| code | Int | | Код ошибки |
| Public методы класса | | | |
| Имя | Принимаемые параметры | Возвращаемое значение | Описание |
| errorText | void | virtual void | Выводит сообщение об ошибке в зависимости от кода |

*Таблица 2.3.8 – Класс InputError (предназначен для обработки исключений ввода)*

|  |  |  |  |
| --- | --- | --- | --- |
| Public методы класса | | | |
| Имя | Принимаемые параметры | Возвращаемое значение | Описание |
| errorText | void | void | Выводит сообщение об ошибке в зависимости от кода |
| input | int min, int max | static int | Проверка введённого значения на принадлежность диапазону [min,max] |

*Таблица 2.3.9 – Класс ListException (предназначен для обработки исключений контейнера)*

|  |  |  |  |
| --- | --- | --- | --- |
| Public методы класса | | | |
| Имя | Принимаемые параметры | Возвращаемое значение | Описание |
| errorText | Void | void | Выводит сообщение об ошибке в зависимости от кода |

*Таблица 2.3.10 – Шаблонная структура Node (предназначена для работы с шаблонным классом- контейнером List)*

|  |  |  |
| --- | --- | --- |
| Public поля класса | | |
| Имя | Тип | Описание |
| data | T | Хранит информацию узла списка |
| next | Node<T>\* | Указывает на следующий элемент списка |
| previous | Node<T>\* | Указывает на предыдующий элемент списка |
| NumberOfSeats | int | Номер узла |

*Таблица 2.3.11 – Шаблонный класс-контейнер List (предназначен для хранения данных различного типа)*

|  |  |  |  |
| --- | --- | --- | --- |
| Protected поля класса | | | |
| Имя | Тип | | Описание |
| Iterator | friend class | | Используется для работы класса-контейнера List |
| head | Node<T>\* | | Указывает на начало списка |
| tail | Node<T>\* | | Указывает на конец списка |
| amount | int | | Количество элементов в списке |
| Public методы класса | | | |
| Имя | Принимаемые параметры | Возвращаемое значение | Описание |
| size | void | int | Возвращает размер списка |
| pushHead | T obj | void | Добавляет элемент в начало списка |
| pushTail | T obj | void | Добавляет элемент в конец списка |
| popHead | void | T data | Удаляет первый элемент из списка и  по умолчанию возвращает данные удалённого элемента |

*Продолжение таблицы 2.3.11*

|  |  |  |  |
| --- | --- | --- | --- |
| popTail | void | T data | Удаляет последний элемент из списка и  по умолчанию возвращает данные удалённого элемента |
| operator[] | int num | T& | Перегруженный оператор индексации |
| begin | void | Node<T>\* | Возвращает указатель на начало списка. Используется при работе с классом Iterator |
| end | void | Node<T>\* | Возвращает указатель на конец списка. Используется при работе с классом Iterator |
| output | void | void | Вывод всех данных из списка на экран |
| sort | int type | void | Сортирует список по определённому признаку |

*Таблица 2.3.12 – Шаблонный класс Iterator (предназначен для работы с классом-контейнером List)*

|  |  |  |  |
| --- | --- | --- | --- |
| Private поля класса | | | |
| Имя | Тип | | Описание |
| current | Node<T>\* | | Указатель на текущий элемент итератора |
| Public методы класса | | | |
| Имя | Принимаемые параметры | Возвращаемое значение | Описание |
| getNode | void | Node<T>\* | Возвращает указатель на текущий элемент итератора |
| operator++ | int i | bool | Сдвигает итератор вниз на 1 |
| operator-- | int i | bool | Сдвигает итератор вверх на 1 |
| operator== | Iterator& iter | bool | Сравнивает  элементы двух объектов класса List<T> |
| operator != | Iterator& iter | bool | Сравнивает  элементы двух объектов класса List<T> |
| operator\*() | void | T& | Перегруженный оператор разадресации |

*Таблица 2.3.13 – Шаблонный класс File (для работы с файлами)*

|  |  |  |  |
| --- | --- | --- | --- |
| Public методы класса | | | |
| Имя | Принимаемые параметры | Возвращаемое значение | Описание |
| openUser | List<T>& list,  const char\* file | void | Открывает файл “users.txt” и считывает из него информацию в список List<User> list |
| openBus | List<T>& list,  const char\* file | void | Открывает файл “bus.txt” и считывает из него информацию в список List<Bus> list |
| openMicrobus | List<T>& list,  const char\* file | void | Открывает файл “microbus.txt” и считывает из него информацию в список List<Microbus> list |
| openElectrobus | List<T>& list,  const char\* file | void | Открывает файл “electrobus.txt” и считывает из него информацию в список List<Electrobus> list |
| openMachine | List<T>& list,  const char\* file | void | Открывает файл “machine.txt” и считывает из него информацию в список List<Machine> list |
| saveUser | List<T>& list,  const char\* file | void | Открывает файл “users.txt” и записывает в него информацию из списка List<User> list |

*Продолжение таблицы 2.3.13*

|  |  |  |  |
| --- | --- | --- | --- |
| saveBus | List<T>& list,  const char\* file | void | Открывает файл “bus.txt” и записывает в него информацию из списка  List<Bus> list |
| saveMicrobus | List<T>& list,  const char\* file | void | Открывает файл “microbus.txt” и записывает в него информацию из списка  List<Microbus> list |
| saveElectrobus | List<T>& list,  const char\* file | void | Открывает файл “electrobus.txt” и записывает в него информацию из списка  List<Electrobus> list |
| saveMachine | List<T>& list,  const char\* file | void | Открывает файл “machine.txt” и записывает в него информацию из списка  List<Machine> list |

*Таблица 2.3.14* **–***Шаблонный класс Algorithm (набор возможных манипуляций с любым типом студентов и пользователями)*

|  |  |  |  |
| --- | --- | --- | --- |
| Public методы класса | | | |
| Имя | Принимаемые параметры | Возвращаемое значение | Описание |
| add | List<T>& list | static void | Используется в операции добавления |
| remove | List<T>& list, vector<T>& returnPrev,  vector<int>& action | static void | Используется в операции удаления элемента в списке |

*Продолжение таблицы 2.3.14*

|  |  |  |  |
| --- | --- | --- | --- |
| update | List<T>&list,  vector<T>&returnPrev,  vector<int>&action,  vector<int>& place | static void | Используется в операции изменения элемента в списке |
| Cancel | List<T>&list,  vector<T>&returnPrev,  vector<int>&action,  vector<int>& place | static void | Используется в операции отмены последнего действия |
| sort | List<T>&list | static void | Используется для сортировки списка |
| registration | void | static void | Используется для регистрации нового пользователя |
| authorithation | void | User& | Используется для авторизации пользователя |

*Таблица 2.3.15 – Шаблонный класс Interface (набор возможных манипуляций с любым типом студентов и пользователями)*

|  |  |  |  |
| --- | --- | --- | --- |
| Public методы класса | | | |
| Имя | Принимаемые параметры | Возвращаемое значение | Описание |
| menu | List<T>& list | static void | Меню работы с информационной системой |
| adminMenu | List<T>& list | static void | Меню работы с информационной системой с правами администратора |
| Authorization | void | User& | Меню для авторизации пользователя |

**3 РАЗРАБОТКА АЛГОРИТМОВ**

**3.1 Разработка схем алгоритмов**

Схема алгоритма записи в файл SaveMicrobus описана в приложении Б.

Схема алгоритма регистрации Registration описана в приложении В.

**3.2 Разработка алгоритмов**

Метод saveBus(List<T>& list, const char \*file) шаблонного класса File:

Шаг 1: Начало.

Шаг 2: Открываем текстовый файл file для записи данных. Если файл не открылся, выводим сообщение “Невозможно открыть файл!” и переходим к шагу 5.

Шаг 3: Контейнер list записываем в файл file.

Шаг 4: Записываем в файл file символ “%”, являющийся условным окончанием записи и предназначенную для повторного считывания данных из файла.

Шаг 5: Конец.

Для алгоритма по шагам рассмотрим методpop\_tail()

шаблонного класса-контейнера List:

Шаг 1: Начало.

Шаг 2: Если указатель head на начало списка list равен nullptr, то возвращаем пустой шаблонный объект класса T и переходим к шагу 8.

Шаг 3: Создаём локальный объект data, который принимает значение tail->data и указатель node на конец списка – tail;

Шаг 4: Если указатели на начало и конец списка не совпадают (т.е в списке больше 1 элемента), то присваиваем tail значение tail->prev, а tail->next приравниваем к nullptr, иначе приравниваем значения head и tail к nullptr;

Шаг 5: Очищаем объём динамической памяти, на который указывает node;

Шаг 6: Уменьшаем параметр списка amount на 1;

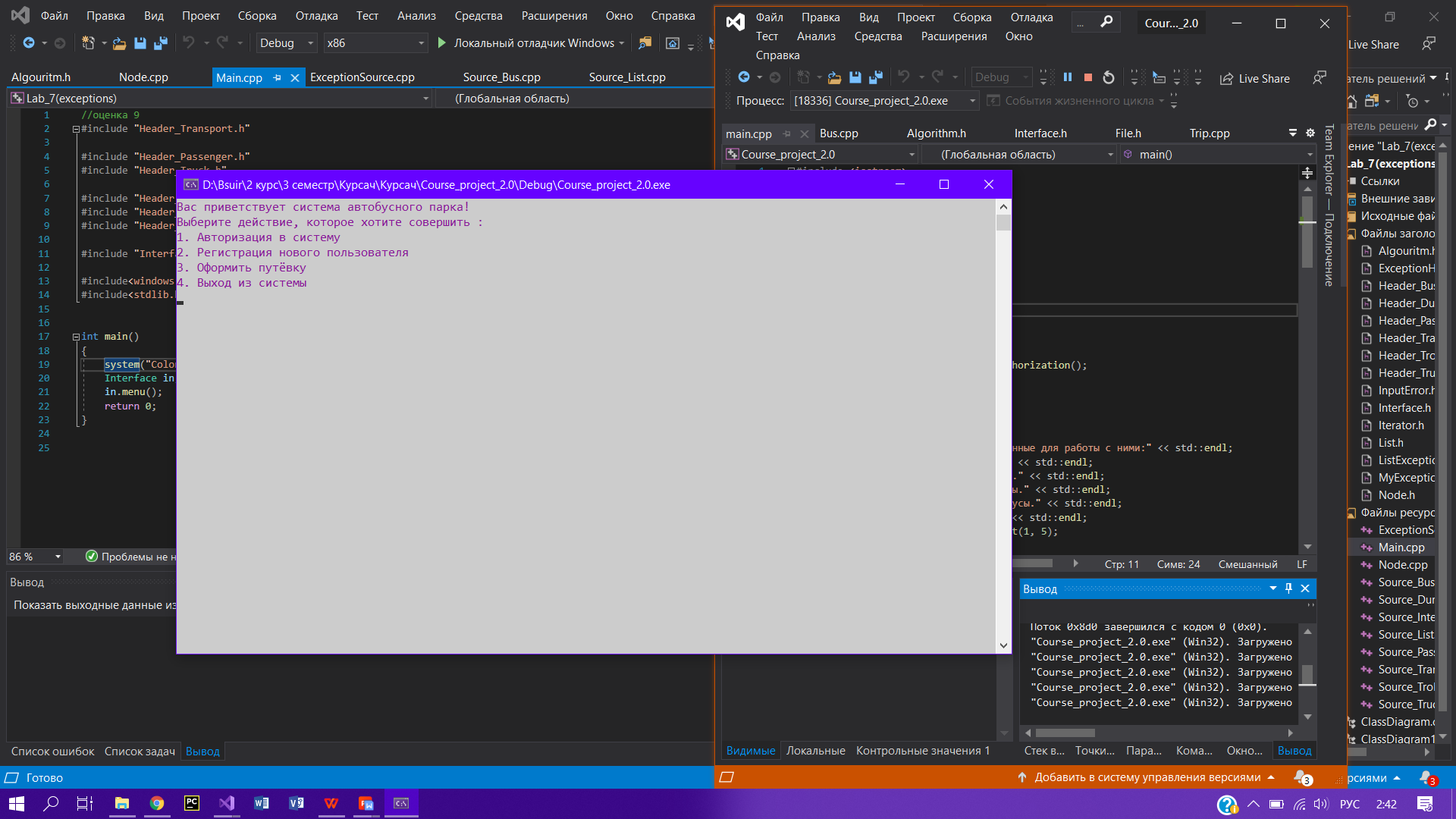
Шаг 7: Возвращаем data;

Шаг 8: Конец.

1. **РЕЗУЛЬТАТЫ РАБОТЫ**

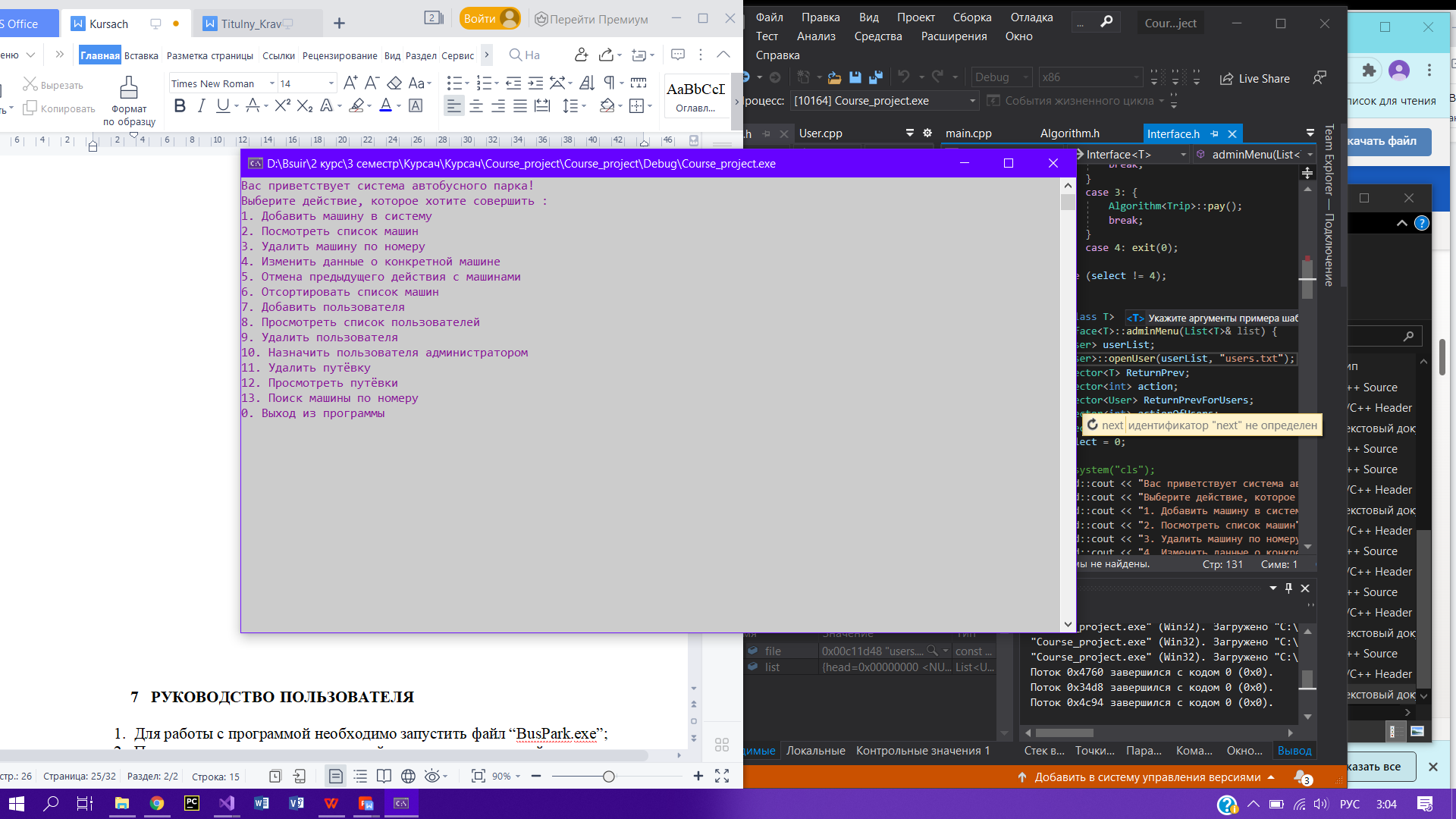
В результате проделанной работы, была получена хорошо отлаженная программа, предоставляющая различные услуги учёта транспортных средств в автопарке.

После запуска программы откроется окно, продемонстрированное на рисунке 1. В этом окне предоставлен выбор: авторизация в систему, регистрация нового пользователя, оформление путёвки или выход из системы .



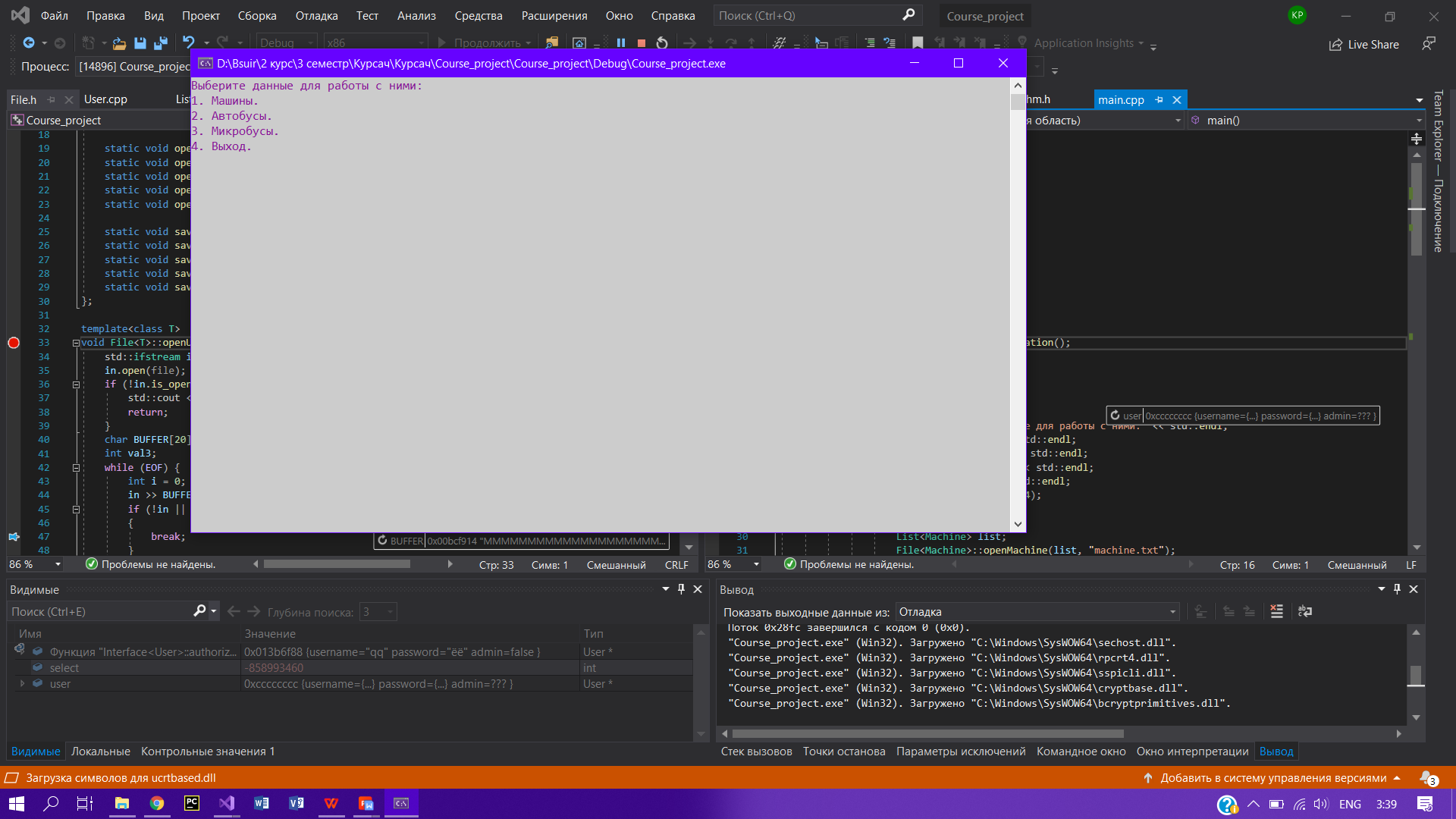
*Рисунок 1*

В случае успешной проверки на права администратора при выборе работы с машинами, пользователь увидит меню, представленное на рис. 2.



*Рисунок 2*

Если пользователь не является администратором, ему предоставляется ограниченный выбор действий, представленный на рисунке 3.



*Рисунок 3*

Программа удобна в использовании и имеет пользовательский интерфейс. После прохождения руководства по использованию выше, применение программы в дальнейшем не составит никаких проблем.

1. **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**
2. Для работы с программой необходимо запустить файл “BusPark.exe”;
3. После запуска программы авторизуйтесь или зарегистрируйтесь и авторизуйтесь;
4. Выберите тип студентов, с которыми хотите работать;
5. Выберите действие, которое необходимо провести со списком данного типа студентов;
6. По окончании работы нажмите 0;
7. Программа успешно сохранила результаты работы в файлах “bus.txt”, ”microbus.txt”, “machine.txt” и ”electrobus.txt”, расположенных в папке с программой.

**ЗАКЛЮЧЕНИЕ**

В результате выполнения курсового проекта было разработано приложение для работы с информационной системой автопарка, позволяющее пользователю вводить различную информацию, выводить ее на экран, изменять, удалять, добавлять, а также, в случае допущения ошибки пользователем, отменять последние действия. Были закреплены знания в области ООП. Использовались среда разработки Visual Studio 2019 и операционная система Windows 10. К достоинствам программы можно отнести простой и понятный интерфейс, что в свою очередь обеспечивает удобство пользования для обычных пользователей. В дальнейшем планируется усовершенствование программы, а именно усовершенствование интерфейса и добавление новых функций, поддержка многопользовательности.

Системные требования:

-Операционная система Windows 98, XP, Vista,7, 8.1, 10;

-Процессор Intel Pentium III 1 GHz;

-Оперативная память 500 MB;

-Свободное место на жестком диске: 5 Мб;

# **СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

[1] Герберт, Ш. Самоучитель C++/Ш. Герберт. Санкт-Петербург 2003г.

[2] Евгений, К. Программирование шахмат и других логических игр/К. Евгений. Санкт-Петербург 2005г.

[3] MSDN – информационный сервис для разработчиков [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/ru-ru/.

[4] [Я. Галовиц](https://oz.by/people/more90125737.html), 2018С++17 STL. Стандартная библиотека шаблонов/[Я. Галовиц](https://oz.by/people/more90125737.html). Санкт-Петербург 2018г.

[5] Страуструп, Б. Программирование. Принципы и практика использования C++/ Б.Страуструп, Вильямс, 2018 г.

[6] Х. М. Дейтел, П. Дж. Дейтел. Как программировать на С++: Пятое издание. Пер. с англ. – М.: ООО “Бином-Пресс”, 2008 г. – 1456 с.: ил.

[7] Н. Литвиненко. Технологии программирования на С++ Win32 API – приложения/Н. Литвиненко. Издательство БХВ, 2006 г.

[8] Д. Харт. Системное программирование в среде Windows/Д. Харт. Издательство “Питер”, 2012 г.

# **ПРИЛОЖЕНИЕ А**

*(обязательное)*

Диаграмма классов

# **ПРИЛОЖЕНИЕ Б**

*(обязательное)*

Схема алгоритма SaveMirobus(List<T>& list, const\* char name)

# **ПРИЛОЖЕНИЕ В**

*(обязательное)*

Схема алгоритма Registration()

# **ПРИЛОЖЕНИЕ Г**

*(обязательное)*

Код программы

**Файл Algorithm.h:**

template <class T>

class Algorithm

{public:

static void registration();

static User\* authorization();

static void add(List<T>& list, std::vector<T>& returnPrev, std::vector<int>& action);

static void remove(List<T> &list, std::vector<T> &returnPrev, std::vector<int> &action);

static void update(List<T> &list, std::vector<T> &returnPrev, std::vector<int> &action, std::vector<int> &place);

static void sort(List<T> &list, std::vector<T> &returnPrev, std::vector<int> &action, std::vector<int> &place);

static void cancel(List<T>& list, std::vector<T>& returnPrev, std::vector<int>& action, std::vector<int>& place);

static void giveAdminRole(List<T>& list);

static void pay();

};

**Файл Algorithm.сpp:**

template<class T>

void Algorithm<T>::add(List<T>& list, std::vector<T>& returnPrev, std::vector<int>& action) {

system("cls");

T tmp;

cin >> tmp;

list.pushTail(tmp);

returnPrev.insert(returnPrev.end(), tmp);

action.insert(action.end(), 1);

system("pause");

}

template<class T>

void Algorithm<T>::remove(List<T> &list, std::vector<T> &returnPrev, std::vector<int>&action) {

system("cls");

if (list.size() == 0) { throw ListException(1); }

else {

cout << "Введите номер элемента для удаления: ";

int num;

num = InputError::Input(1, list.size());

T tmp = list[num - 1];

returnPrev.insert(returnPrev.end(), tmp);

action.insert(action.end(), 2);

list.pop(num);

}

system("pause");

}

template<class T>

void Algorithm<T>::update(List<T> &list, std::vector<T> &returnPrev, std::vector<int> &action, std::vector<int> &place) {

system("cls");

if (!list.size()) {

throw ListException(1);

}

list.output();

cout << endl;

cout << "Введите номер изменяемой машины : ";

int tempNum = InputError::Input(1, list.size());

T tmp;

cin >> tmp;

returnPrev.insert(returnPrev.end(), list[tempNum - 1]);

action.insert(action.end(), 3);

place.insert(place.end(), tempNum);

list[tempNum - 1] = tmp;

system("pause");

}

template<class T>

void Algorithm<T>::sort(List<T> &list, std::vector<T> &returnPrev, std::vector<int> &action, std::vector<int> &place) {

system("cls");

if (!list.size()) {throw ListException(1); }

list.output();

cout << std::endl;

cout << "Выберите по какому признаку отсортировать:" << endl;

cout << "1. По ID по возрастанию." << endl;

cout << "2. По ID по убыванию." << endl;

cout << "3. По оплате по возрастанию." << endl;

cout << "4. По оплате по убыванию." << endl;

int type = InputError::Input(1, 4);

list.sort(type);

list.output();

}

template<class T>

void Algorithm<T>::cancel(List<T>& list, std::vector<T>& returnPrev, std::vector<int>& action, std::vector<int>& place) {

if (!action.size()) {

system("cls");

throw ListException(2);

}

switch (action.back()) {

case 1: {

list.popTail();

action.erase(action.end() - 1);

returnPrev.erase(returnPrev.end() - 1);

break;

}

case 2: {

list.pushTail(returnPrev.back());

action.erase(action.end() - 1);

returnPrev.erase(returnPrev.end() - 1);

break;

}

case 3: {

list[place.back() - 1] = returnPrev.back();

place.erase(place.end() - 1);

action.erase(action.end() - 1);

returnPrev.erase(returnPrev.end() - 1);

break;

}

}

}

template<class T>

User\* Algorithm<T>::authorization() {

List<User> users;

File<User>::openUser(users, "users.txt");

char BUFFER[20], c;

string username, password;

cout << "Введите логин: ";

cin >> username;

cout << "Введите пароль: ";

int i = 0;

while ((c = \_getch()) != '\r') {

\_putch('\*');

BUFFER[i] = c;

i++;

}

BUFFER[i] = '\0';

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

BUFFER[j] = 6 \* BUFFER[j] + (BUFFER[j] / 6);

}

password = string(BUFFER);

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

if (users[i].getUsername() == username && users[i].getPassword() == password)

{ return new User(users[i]); }

}

cout << "Неверный пользователь или пароль!" << endl;

return nullptr;

}

template<class T>

void Algorithm<T>::registration() {

List<User> users;

File<User>::openUser(users, "users.txt");

User newUser;

cin >> newUser;

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

if (users[i].getUsername() == newUser.getUsername() && users[i].getPassword() == newUser.getPassword() && newUser.isAdmin() != users[i].isAdmin()) {

cout << "Данный пользователь уже существует!" << endl;

return;

}

}

users.pushTail(newUser);

File<User>::saveUser(users, "users.txt");

}

template<class T>

void Algorithm<T>::giveAdminRole(List<T> &list) {

list.output();

cout << std::endl;

cout << "Введите номер пользователя для предоставления прав администратора : ";

int tempNum = InputError::Input(1, list.size());

list[tempNum - 1].setAdmin(1);

}

template<class T>

void Algorithm<T>::pay() {

cout << "Заполните форму!" << endl;

Trip trip;

cin >> trip;

List<Trip> tripList;

File<Trip>::openTrip(tripList, "trip.txt");

tripList.pushTail(trip);

File<Trip>::saveTrip(tripList, "trip.txt");

cout << "Форма заполнена успешно!" << endl;

}

**Файл Bus.h:**

class Bus : public Machine {

protected:

int status;

bool type;

public:

Bus(int sum = 0, int id = 0, int numberOfSeats = 0, int day = 0,

const char \*features = "", int status = 0, bool type = false);

int getStatus() const;

void setStatus(int status);

bool isType() const;

void setType(bool type);

friend istream &operator>>(istream &is, Bus &bus);

friend ostream &operator<<(ostream &os, const Bus &bus);

};

**Файл Bus.cpp:**

Bus::Bus(int sum, int id, int numberOfSeats, int day, const char \*features,

int status, bool type) : Machine(sum, id, numberOfSeats, day, features), status(status), type(type) {}

int Bus::getStatus() const { return status;}

void Bus::setStatus(int status) { Bus::status = status;}

bool Bus::isType() const {return type;}

void Bus::setType(bool type) { Bus::type = type;}

ostream &operator<<(ostream &os, const Bus &bus) {

os << "Статус (1. На базе. 2. В рейсе. 3. В ремонте. 4. Списан.): " << bus.status << endl << "Тип автобуса (0 - Мягкий, 1 - Жёсткий): " << bus.type << endl << static\_cast<const Machine &>(bus);

return os;

}

istream &operator>>(istream &is, Bus &bus) {

cout << "Введите статус автобуса (1. На базе. 2. В рейсе. 3. В ремонте. 4. Списан.): ";

bus.status = InputError::Input(1, 4);

cout << "Введите тип автобуса (0 - Мягкий, 1 - Жёсткий): ";

bus.type = InputError::Input(0, 1);

is >> dynamic\_cast<Machine&>(bus);

return is;

}

**Файл Electrobus.cpp:**

class Electrobus : public Bus

{

private:

bool harmonic;

bool ecological;

bool wirelessCharger;

public:

explicit Electrobus(bool harmonic = false, bool ecological = false,

bool wirelessCharger = false, int sum = 0, int id = 0, int numberOfSeats = 0, int day = 0,const char\* features = "", int status = 0, bool type = false);

bool isHarmonic() { return harmonic; }

bool isEcological() { return ecological; }

bool isWirelessCharger() { return wirelessCharger; }

friend istream& operator>>(istream& is, Electrobus& electrobus);

friend ostream& operator<<(ostream& os, const Electrobus& electrobus);

};

**Файл Electrobus.cpp:**

Electrobus::Electrobus(bool harmonic, bool ecological, bool wirelessCharger, int sum, int id, int numberOfSeats, int day, const char\* features, int status, bool type) : Bus(sum, id, numberOfSeats, day, features, status, type), harmonic(harmonic), ecological(ecological), wirelessCharger(wirelessCharger) {}

ostream& operator<<(ostream& os, const Electrobus& electrobus) {

os << static\_cast<const Bus&>(electrobus);

os << "Дополнительно (1 - Есть. 0 - Нет)" << endl;

os << "Наличие гармошки: " << electrobus.harmonic << endl;

os << "Экологический: " << electrobus.ecological << endl;

os << "Наличие беспроводной зарядки: " << electrobus.wirelessCharger << endl;

return os;

}

istream& operator>>(istream& is, Electrobus& electrobus) {

is >> dynamic\_cast<Bus&>(electrobus);

cout << "Дополнительно (1 - Есть. 0 - Нет):" << endl;

cout << "Есть ли гармошка: ";

electrobus.harmonic = InputError::Input(0, 1);

cout << "Является ли экологическим: ";

electrobus.ecological = InputError::Input(0, 1);

cout << "Есть ли беспроводная зарядка: ";

electrobus.wirelessCharger = InputError::Input(0, 1);

return is;

}

**Файл File.h:**

template <class T>

class File {

public:

File() = default;

~File() = default;

static void openUser(List<T>& list, const char \*file);

static void openMachine(List<T>& list, const char \*file);

static void openBus(List<T>& list, const char\* file);

static void openElectrobus(List<T>& list, const char \*file);

static void openMicrobus(List<T>& list, const char \*file);

static void openTrip(List<T>& list, const char \*file);

static void saveTrip(List<T>& list, const char \*file);

static void saveUser(List<T>& list, const char \*file);

static void saveMachine(List<T>& list, const char \*file);

static void saveBus(List<T>& list, const char\* file);

static void saveElectrobus(List<T>& list, const char \*file);

static void saveMicrobus(List<T>& list, const char \*file);

};

**Файл File.cpp:**

template <class T>

void File<T>::openUser(List<T>& list, const char\* file)

{

ifstream in;

in.open(file);

if (!in.open())

{

cout << "Невозможно открыть файл!" << endl;

return;

}

char BUFFER[20];

int val3;

while (EOF)

{

int i = 0;

in >> BUFFER[i];

if (!in || BUFFER[i++] == '%') { break;}

while ((BUFFER[i] = in.get()) != '|' && BUFFER[i] != '\n') {i++;}

BUFFER[i] = '\0';

string val1(BUFFER);

i = 0;

while ((BUFFER[i] = in.get()) != '|' && BUFFER[i] != '\n') {i++;}

BUFFER[i] = '\0';

string val2(BUFFER);

in >> val3;

T t(val1, val2, val3);

list.pushTail(t);

}

in.close();

}

template <class T>

void File<T>::openMachine(List<T>& list, const char\* file)

{

istream in;

in.open(file, ios::in);

if (!in.open())

{

cout << "Невозможно открыть файл" << endl;

return;

}

char BUFFER[50];

char val5[50];

int val1, val2, val3, val4;

while(EOF)

{

int i = 0;

in >> BUFFER[i];

if (!in || BUFFER[i++] == '%') { break;}

while ((BUFFER[i] = in.get()) != '|' && BUFFER[i] != '\n') { i++;}

BUFFER[i] = '\0';

strcpy\_s(val5, BUFFER);

in.get();

in >> val1 >> val2 >> val3 >> val4;

Machine machine(val1, val2, val3, val4, val5);

list.pushTail(machine);

}

in.close();

}

template<class T>

void File<T>::openBus(List<T>& list, const char\* file)

{

istream in;

in.open(file);

is(!in.is\_open())

{

cout << "Невозможно открыть файл" << endl;

return;

}

char BUFFER[50];

char val5[50];

int val1, val2, val3, val4, val6;

bool val7;

while (EOF)

{

int i = 0;

in >> BUFFER[i];

if (!in || BUFFER[i++] == '%') {break;}

while ((BUFFER[i] == in.get()) != '|' && BUFFER[i] != '\n') {i++;}

BUFFER[i] = '\0';

strcpy\_s(val5, BUFFER);

in.get();

in >> val1 >> val2 >> val3 >> val4 >> val6 >> val7;

Bus bus(val1, val2, val3, val4, val5, val6, val7);

list.pushTail(bus);

}

in.close();

}

template <class T>

void File<T>::openElectrobus(List<T>& list, const char\* file)

{

istream in;

in.open(file);

if (!in.is\_open())

{

cout << "Невовзможно открыть файл" << endl;

return;

}

char BUFFER[50];

char val5[50];

int val1.val2, val3, val4, val6;

bool val7, val8, val9, val10;

while(EOF)

{

int i = 0;

in >> BUFFER[i];

if (!in || BUFFER[i++] == '%') { break; }

while ((BUFFER[i] = in.get()) != '|' && BUFFER[i] != '\n') { i++;}

BUFFER[i] = '\0';

strcpy\_s(val5, BUFFER);

in.get();

in >> val1 >> val2 >> val3 >> val4 >> val5 >> val6 >> val7 >> val8 >> val9 >> val10;

Electrobus bus(val8, val9, val10, val1, val2, val3, val4, val5, val6, val7);

list.pushTail(bus);

}

in.close();}

template <class T>

void File<T>::openMicrobus(List<T>& list, const char\* file)

{

ifstream in;

in.open(file, ios::in);

if (!in.is\_open())

{

cout << "Невозможно открыть файл" << endl;

return;

}

int val1, val3, val3, val4;

while (EOF | in.failbit)

{

in >> val1;

if (in.failbit)

{

in.clear();

break;

}

in >> val2 >> val3 >> val4;

Microbus microbus(val1, val2, val3, val4);

list.pushTail(microbus);

in.get();

}

in.close();

}

template <class T>

void File<T>::saveUser(List<T>& list, const char\* file)

{

ofstream on;

on.open(file, ios::out);

if (!on.is\_open())

{

cout << "Невозможно открыть файл" << endl;

return;

}

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

{

on << list[i].getUsrename() << "|" << list[i].hetPassword() << "|" << list[i].isAdmin();

on << endl;

}

on << "%";

on.close();

}

template <class T>

void File<T>::saveMachine(List<T>& list, const char\* file)

{

ofstream on;

on.open(file, ios\_base::out);

if (!on.is\_open())

{

cout << "Невозможно открыть файл" << endl;

return;

}

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

{

on << list[i].getFeatures() << "|" << list[i].getSum() << " " << list[i].getId() << " " <<

list[i].getNumberOfSeats() << " " << list[i].getDay();

on << endl;

}

on << "%";

on.close();

}

template<class T>

void File<T>::saveBus(List<T> &list, const char \*file) {

ofstream on;

on.open(file, ios\_base::out);

if (!on.is\_open()) {

cout << "Невозможно открыть файл!" << endl;

return;

}

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

on << list[i].getFeatures() << "| " << list[i].getSum() << " " << list[i].getId() << " " << list[i].getNumberOfSeats() << " " << list[i].getDay() <<

" " << ist[i].getStatus() << " " << list[i].isType();

on << endl;

}

on << "%";

on.close();

}

template<class T>

void File<T>::saveElectrobus(List<T>& list, const char\* file) {

ofstream on;

on.open(file, ios\_base::out);

if (!on.is\_open()) {

cout << "Невозможно открыть файл!" << endl;

return;

}

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

on << list[i].getFeatures() << "| " << list[i].getSum() << " " << list[i].getId()

<< " " << list[i].getNumberOfSeats() << " " << list[i].getDay() << " "

<< list[i].getStatus() << " " << list[i].isType() << " "

<< list[i].isHarmonic() << " " << list[i].isEcological() << " "

<< list[i].isWirelessCharger();

on << endl;

}

on << "%";

on.close();

}

template<class T>

void File<T>::saveMicrobus(List<T> &list, const char \*file) {

ofstream on;

on.open(file, ios\_base::out);

if (!on.is\_open()) {

cout << "Невозможно открыть файл!" << endl;

return;

}

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

on << list[i].getSum() << " " << list[i].getId() << " " <<

list[i].getBrand() << " " << list[i].getNumberOfCompletedTrips();

on << endl;

}

on << "%";

on.close();

}

template<class T>

void File<T>::openTrip(List<T> &list, const char \*file) {

ifstream in;

in.open(file);

if (!in.is\_open()) {

cout << "Невозможно открыть файл!" << endl;

return;

}

rewind(stdin);

string val1;

int val2, val3, val4;

while (EOF) {

getline(in, val1);

if (!in || val1.empty() || val1 == "%") {

in.clear();

break;

}

in >> val2 >> val3 >> val4;

Trip trip(val1, val2, val3, val4);

list.pushTail(trip);

in.get();

}

in.close();

}

template<class T>

void File<T>::saveTrip(List<T> &list, const char \*file) {

ofstream on;

on.open(file, ios\_base::out);

if (!on.is\_open()) {

cout << "Невозможно открыть файл!" << endl;

return;

}

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

on << list[i].getDestination() << "\n" << list[i].getWay() << " " << list[i].getDay() << " " << list[i].getNumber();

on << endl;

}

on << "%";

on.close();

}

**Файл InputError.h:**

namespace Error {

class InputError : public MyException {

public:

InputError(int code = 0) : MyException(code) {};

void ErrorText() override;

static int Input(int min, int max);

static int InputNumber();

static char\* inputStringWithoutNumbers();

};

}

**Файл InputError.cpp:**

void InputError::ErrorText()

{

cout << "Ошибка ввода (код " << this->code << "): ";

switch (this->code)

{

case 1:

{

cout << "Введён недопустимый символ." << endl;

cout << "Ввод любых символов, кроме цифр, запрещён." << endl;

break;

}

case 2:

{

cout << "Вы ввели символы вместо числа либо превышен предел по вводу чисел!" << endl;

break;

}

default:

cout << "Повторите ввод" << endl;

break;

}

}

int InputError::Input(int min, int max)

{

bool flag = true;

int value;

do{

try{

rewind(stdin);

cin >> value;

flag = true;

if (!cin || cin.peek() != '\n'){

flag = false;

throw InputError(2);

}

if (value<min || value>max){

flag = false;

throw "Limit error";

}

}

catch (InputError e){

rewind(stdin);

e.ErrorText();

}

catch (...){

rewind(stdin);

cout << "Введено некорректное значение. Необходимо ввести значение в диапазоне (" << min << "," << max << ")." << endl;

}

} while (!flag);

return value;

}

int InputError::InputNumber()

{

int input;

try{

rewind(stdin);

cin >> input;

if (!cin || cin.peek() != '\n') { throw InputError(2); }

}

catch (InputError e){

e.ErrorText();

rewind(stdin);

input = InputNumber();

}

rewind(stdin);

return input;

}

char\* InputError::inputStringWithoutNumbers()

{

rewind(stdin);

char input;

char\* buffer = new char[20];

int i = 0;

bool flag = true;

while (flag) {

try {

input = cin.get();

if (input != '\n' && (input < 'a' || input > 'z') && (input < 'A' || input > 'Z')) {

throw (InputError(4));

}

if (input == '\n') {

input = '\0';

flag = false;

}

buffer[i] = input;

i++;

}

catch (InputError e)

{

rewind(stdin);

e.ErrorText();

}

}

return buffer;

}

**Файл Interface.h:**

template<class T>

class Interface {

public:

static void menu(List<T>& list);

static void adminMenu(List<T>& list);

static User\* authorization();

};

**Файл Interface.cpp:**

template<class T>

void Interface<T>::menu(List<T>& list) {

std::vector<T> ReturnPrev;

std::vector<int> action;

std::vector<int> place;

int select = 0;

do {

system("cls");

cout << "Вас приветствует система автобусного парка!" << endl;

cout << "Выберите действие, которое хотите совершить : " << endl;

cout << "1. Добавить машину в систему" << endl;

cout << "2. Посмотреть список машин" << endl;

cout << "3. Удалить последнюю машину из системы" << endl;

cout << "4. Изменить данные о конкретной машине" << endl;

cout << "5. Отмена предыдущего действия с машинами" << endl;

cout << "6. Отсортировать список машин" << endl;

cout << "0. Выход из программы" << endl;

try {

select = InputError::Input(0, 6);

switch (select) {

case 1:

{

system("cls");

Algorithm<T>::add(list, ReturnPrev, action);

break;

}

case 2:

{

system("cls");

if (list.size() == 0) {

throw ListException(1);

} else {

list.output();

cout << endl;

system("pause");

}

break;

}

case 3:

{

system("cls");

Algorithm<T>::remove(list, ReturnPrev, action);

break;

}

case 4:

{

system("cls");

Algorithm<T>::update(list, ReturnPrev, action, place);

break;

}

case 5:

{

system("cls");

Algorithm<T>::cancel(list, ReturnPrev, action, place);

break;

}

case 6:

{

system("cls");

Algorithm<T>::sort(list, ReturnPrev, action, place);

break;

}

}

}

catch (ListException e) {

e.ErrorText();

system("pause");

cin.sync();

cin.clear();

rewind(stdin);

}

} while (select != 0);

}

template<class T>

User\* Interface<T>::authorization() {

int select = 0;

do {

system("cls");

cout << "Вас приветствует система автобусного парка!" << endl;

cout << "Выберите действие, которое хотите совершить : " << endl;

cout << "1. Авторизация в систему" << endl;

cout << "2. Регистрация нового пользователя" << endl;

cout << "3. Оформить путёвку" << endl;

cout << "4. Выход из системы" << endl;

select = InputError::Input(1, 4);

switch (select) {

case 1: {

User \*currentUser = Algorithm<User>::authorization();

if (currentUser != nullptr) { return currentUser; }

break;

}

case 2: {

system("cls");

Algorithm<User>::registration();

break;

}

case 3: {

Algorithm<Trip>::pay();

break;

}

case 4: exit(0);

}

} while (select != 4);

}

template<class T>

void Interface<T>::adminMenu(List<T>& list) {

List<User> userList;

File<User>::openUser(userList, "users.txt");

std::vector<T> ReturnPrev;

std::vector<int> action;

std::vector<User> ReturnPrevForUsers;

std::vector<int> actionOfUsers;

std::vector<int> place;

int select = 0;

do {

system("cls");

cout << "Вас приветствует система автобусного парка!" << endl;

cout << "Выберите действие, которое хотите совершить : " << endl;

cout << "1. Добавить машину в систему" << endl;

cout << "2. Посмотреть список машин" << endl;

cout << "3. Удалить последнюю машину из системы" << endl;

cout << "4. Изменить данные о конкретной машине" << endl;

cout << "5. Отмена предыдущего действия с машинами" << endl;

cout << "6. Отсортировать список машин" << endl;

cout << "7. Добавить пользователя" << endl;

cout << "8. Просмотреть список пользователей" << endl;

cout << "9. Удалить пользователя" << endl;

cout << "10. Назначить пользователя администратором" << endl;

cout << "11. Удалить путёвку" << endl;

cout << "12. Просмотреть путёвки" << endl;

cout << "13. Поиск машины по номеру" << endl;

cout << "0. Выход из программы" << endl;

try {

select = InputError::Input(0, 16);

switch (select) {

case 1:

{

system("cls");

Algorithm<T>::add(list, ReturnPrev, action);

break;

}

case 2:

{

system("cls");

if (list.size() == 0) { throw ListException(1); }

else {

list.output();

cout << endl;

system("pause");

}

break;

}

case 3:

{

system("cls");

Algorithm<T>::remove(list, ReturnPrev, action);

break;

}

case 4:

{

system("cls");

Algorithm<T>::update(list, ReturnPrev, action, place);

break;

}

case 5:

{

system("cls");

Algorithm<T>::cancel(list, ReturnPrev, action, place);

break;

}

case 6:

{

system("cls");

Algorithm<T>::sort(list, ReturnPrev, action, place);

break;

}

case 7: {

system("cls");

Algorithm<User>::add(userList, ReturnPrevForUsers, actionOfUsers);

File<User>::saveUser(userList, "users.txt");

break;

}

case 8: {

system("cls");

if (userList.size() == 0) {throw ListException(1); }

else {

userList.output();

cout << endl;

system("pause");

}

break;

}

case 9: {

system("cls");

if (userList.size() == 0) {throw ListException(1);}

else {

userList.output();

cout << endl;

cout << "Введите номер пользователя для удаления: ";

int tempNum = InputError::Input(1, userList.size());

userList.pop(tempNum - 1);

File<User>::saveUser(userList, "users.txt");

system("pause");

}

break;

}

case 10: {

system("cls");

Algorithm<User>::giveAdminRole(userList);

File<User>::saveUser(userList, "users.txt");

break;

}

case 11:

{

List<Trip> tripList;

File<Trip>::openTrip(tripList, "trip.txt");

tripList.popTail();

File<Trip>::saveTrip(tripList, "trip.txt");

break;

}

case 12:

{

List<Trip> tripList;

File<Trip>::openTrip(tripList, "trip.txt");

cout << "№ | Пункт назначения |Дальность пути|День\n";

tripList.output();

break;

}

case 13:

{

if (list.size() == 0) {throw ListException(1);}

else {

list.output();

cout << endl << "Введите номер для поиска: ";

int id = InputError::Input(100000, 999999);

bool flag = false;

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

if (list[i].getId() == id) {

flag = true;

cout << list[i];

}

}

if (!flag) {cout << "Ничего не найдено!" << endl;}

system("pause");

}

break;

}

}

}

catch (ListException e) {

e.ErrorText();

system("pause");

cin.sync();

cin.clear();

rewind(stdin);

}

} while (select != 0);

}

**Файл List.h:**

template <class T>

struct Node {

T data;

Node<T>\* previous;

Node<T>\* next;

};

template <class T>

class List {

private:

Node<T> \*head = nullptr;

Node<T> \*tail = nullptr;

int amount = 0;

public:

List() {};

~List();

int size() { return this->amount; }

void pushHead(T obj);

void push(T obj, int num);

void pushTail(T input\_object);

T popHead();

T pop(int num);

T popTail();

T& operator[](int num);

Node<T>\* begin();

Node<T>\* end();

void sort(int type);

void output();

void deleteAll();

};

**Файл List.cpp:**

template <class T>

List<T>::~List() { while (this->head) { this->popHead(); } }

template<class T>

void List<T>::pushHead(T obj)

{

if (head == nullptr){

head = new Node<T>;

head->data = obj;

head->next = nullptr;

head->previous = nullptr;

tail = head;

amount++;

}

else{

auto\* node = new Node<T>;

node->data = obj;

node->next = head;

node->previous = nullptr;

head->previous = node;

head = node;

amount++;

}

}

template<class T>

void List<T>::push(T obj, int num)

{

Node<T>\* next = this->operator[](num);

Node<T>\* prev = this->operator[](num - 1);

auto\* node = new Node<T>;

node->data = obj;

node->next = next;

next->previous = node;

node->previous = prev;

prev->next = node;

amount++;

}

template<class T>

void List<T>::pushTail(T input\_object)

{

if (head == nullptr){

head = new Node<T>;

head->data = input\_object;

head->next = nullptr;

head->previous = nullptr;

tail = head;

amount++;

}

auto\* node = new Node<T>;

node->data = input\_object;

node->next = nullptr;

node->previous = tail;

tail->next = node;

tail = node;

amount++;

}

template<class T>

T List<T>::popHead()

{

if (!(head)) return T();

T data = head->data;

Node<T>\* node = head;

if (head != tail){

head = head->next;

head->previous = nullptr;

Node<T>\* tmp = head;

}

else{ head = tail = nullptrl;}

delete node;

amount--;

return data;

}

template<class T>

T List<T>::pop(int num)

{

if (num == 0)return popHead();

if (num == amount - 1) return popTail();

Node<T>\* curr = head;

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

curr = curr->next;

Node<T>\* prev = curr->previous;

Node<T>\* next = curr->next;

T data = curr->data;

amount--;

prev->next = next;

next->previous = prev;

delegate curr;

return data;

}

template<class T>

T List<T>::popTail()

{

if (!(head)) return T();

T data = tail->data;

Node<T>\* node = tail;

if (tail != head){

tail = tail->previous;

tail->next = nullptr;

}

Else {head = tail = nullptr;}

delete node;

amount--;

return data;

}

template<class T>

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

{

Node<T>\* curr = head;

if (num < 0 || num > amount) return curr->data;

for (int i = 0; i < num; i++) { curr = curr->next;}

return curr->data;

}

template <class T>

void List<T>::sort(int type)

{

Node<T>\* tmp = this->head;

Node<T>\* another = this->tail;

switch (type)

{

case 1:

{

for (tmp; tmp->next != nullptr; tmp = tmp->next)

{

for (another; another != tmp; another = another->previous)

{

if (tmp->data.getId() > another->data.getUd())

{

auto\* buf1 = new Node<T>;

buf1->data = tmp->data;

tmp->data = another->data;

another->data = buf1->data;

}

}

}

break;

}

case 2:

{

for (tmp; tmp->next != nullptr; tmp = tmp->next)

{

for (another; another != tmp; another = another->previous)

{

if (tmp->data.getId() < another->data.getUd())

{

auto\* buf1 = new Node<T>;

buf1->data = tmp->data;

tmp->data = another->data;

another->data = buf1->data;

}

}

}

break;

}

case 3:

{

for (tmp; tmp->next != nullptr; tmp = tmp->next)

{

for (another; another != tmp; another = another->previous)

{

if (tmp->data.getSum() > another->data.getSum())

{

auto\* buf1 = new Node<T>;

buf1->data = tmp->data;

tmp->data = another->data;

another->data = buf1->data;

}

}

}

break;

}

case 4:

{

for (tmp; tmp->next != nullptr; tmp = tmp->next)

{

for (another; another != tmp; another = another->previous)

{

if (tmp->data.getSum() < another->data.getSum())

{

auto\* buf1 = new Node<T>;

buf1->data = tmp->data;

tmp->data = another->data;

another->data = buf1->data;

}

}

}

break;

}

default:

break;

}

}

template <class T>

void List<T>::deleteAll()

{

while (this->head)

{

this->popHead();

}

}

**Файл Iterator.h:**

template <class T>

class Iterator

{

Node<T>\* current;

public:

Iterator() { current = nullptr; }

Iterator(Iterator& iter) { return current; }

~Iterator() = default;

Node<T>\* getNode() { return current; }

List<T> begin();

List<T> end();

bool operator ++(int i);

bool operator --(int i);

bool operator ==(Iterator& iter);

bool operator !=(Iterator& iter);

T& operator\*() { return current->data; }

};

**Файл Iterator.cpp:**

template <class T>

bool Iterator<T>:: operator ++(int i)

{

if (current == nullptr) return false;

current = current->next;

return true;

}

template <class T>

bool Iterator<T>:: operator --(int i)

{

if (current->next == nullptr) return false;

current = current->prev;

return true;

}

template <class T>

bool Iterator<T>::operator ==(Iterator& iter)

{

if (current == nullptr && iter.current == nullptr)

return true;

if (current == nullptr || iter.current == nullptr)

return false;

if (current->data == iter.current->data && current->next == iter.current->next && current->previous == iter.current->previous)

return true;

return false;

}

template <class T>

bool Iterator<T>::operator !=(Iterator& iter)

{

if (current == nullptr && iter.current == nullptr)

return false;

if (current == nullptr || iter.current == nullptr)

return true;

if (current->data == iter.current->data && current->next == iter.current->next && current->previous == iter.current->previous)

return false;

return true;

}

template<class T>

List<T> Iterator<T>::begin()

{

return head;

}

template<class T>

List<T> Iterator<T>::end()

{

if (tail != nullptr) return tail->next;

return tail;

}

**Файл ListExeption.h:**

class ListExeption :MyExeption

{

public:

explicit ListExeption(int code = 0) :MyExeption(code) {};

void ErrorText() override;

};

**Файл ListExeption.cpp:**

void ListExeption::ErrorText()

{

cout << "Ошибка (код " << code << "): " << endl;

switch (code)

{

case 1:

{

cout << "Список пуст" << endl;

break;

}

case 2:

{

cout << "Невозможно отменить действие.";

cout << "Все возможные последние действия со студентами отменены." << endl;

break;

}

default:

break;

}

}

**Файл Machine.h:**

class Machine : public Ticket

{

protected:

int numberOfSeats;

int day;

char features[50];

public:

Machine(int sum = 0, int id = 0, int numberOfSeats = 0, int day = 0, const char\* featuers = "");

int getNumberOfSeats() const;

void setNumberOfSeats(int numberOfSeats);

int getDay() const;

void setDay(int day);

const char\* getFeatures() const;

void setFeatures(const char\* features);

friend istream& operator >> (istream& is, Machine& machine);

friend ostream& operator << (ostream& os, const Machine& machine);

};

**Файл Machine.cpp:**

Machine::Machine(int sum, int id, int numberOfSeats, int day, const char\* features) :

Ticket(sum, id), numberOfSeats(numberOfSeats), day(day)

{

strcpy\_s(this->features, features);

}

int Machine::getNumberOfSeats() const { return numberOfSeats;}

void Machine::setNumberOfSeats(int numberOfSeats) {this->numberOfSeats = numberOfSeats;}

int Machine::getDay() const {return day;}

void Machine::setDay(int day) {this->day = day;}

const char\* Machine::getFeatures() const {return features;}

void Machine::setFeatures(const char\* features) {strcpy\_s(Machine::features, features);}

ostream& operator << (ostream& os, const Machine& machine)

{

os << "Количество мест: " << machine.numberOfSeats << endl

<< "Срок эксплуатвции: " << machine.day << endl <<"Особенности использования:" << machine.features << endl;

os << static\_cast <const Ticket&>(machine);

return os;

}

istream& operator >> (istream& is, Machine& machine)

{

cout << "Введите количество мест: ";

machine.numberOfSeats = InputError::Input(0, 1000);

cout << "Введите срок эксплуатации: ";

machine.day = InputError::Input(0, 100);

rewind(stdin);

cout << "Введите особенности использования: ";

is >> machine.features;

is >> dynamic\_cast<Ticket&>(machine);

return is;

}

**Файл Microbus.h:**

class Microbus:public Ticket

{

private:

int brand;

int numberOfCompletedTrips;

public:

Microbus(int sum = 0, int id = 0, int brand = 0, int numberOfCompletedTrips = 0);

int getBrand() const;

void setBrand(int brand);

int getNumberOfCompletedTrips() const;

void setNumberOfCompletedTrips(int numberOfCompletedTrips);

friend istream& operator>>(istream& is, Microbus& microbus);

friend ostream& operator<<(ostream& os, const Microbus& microbus);

};

**Файл Microbus.cpp:**

Microbus::Microbus(int sum, int id, int brand, int numberOfCompletedTrips) : Ticket(sum, id), brand(brand), numberOfCompletedTrips(numberOfCompletedTrips) {}

int Microbus::getBrand() const { return brand;}

void Microbus::setBrand(int brand) { this->brand = brand; }

int Microbus::getNumberOfCompletedTrips() const { return numberOfCompletedTrips; }

void Microbus::setNumberOfCompletedTrips(int numberOfCompletedTrips)

{

this->numberOfCompletedTrips = numberOfCompletedTrips;

}

ostream& operator << (ostream& os, const Microbus& microbus)

{

os << "№ бренда:" << microbus.brand << endl << "Количество выполненных поездок: "

<< microbus.numberOfCompletedTrips << endl;

os << static\_cast<const Ticket&>(microbus);

return os;

}

istream& operator >>(istream& is, Microbus& microbus)

{

cout << "Введите номер бренда: ";

microbus.brand = InputError::Input(1, 999);

cout << "Введите количество поездок, выполненных микроавтобусом: ";

microbus.numberOfCompletedTrips = InputError::Input(0, 1000);

is >> dynamic\_cast<Ticket&>(microbus); return is;

}

**Файл MyExeprion.h:**

class MyExeption

{

protected:

int code;

public:

explicit MyExeption(int code = 0) { this->code = code; }

virtual void ErrorText() {};

};

**Файл Ticket.h:**

class Ticket

{

protected:

int sum;

int id;

public:

Ticket(int sum = 0, int id = 0);

int getSum() const;

void setSum(int sum);

int getId() const;

void setId(int id);

friend istream& operator>>(istream& is, Ticket& ticket);

friend ostream& operator << (ostream& os, const Ticket& ticket);

};

**Файл Ticket.cpp:**

Ticket::Ticket(int sum, int id) : sum(sum), id(id) {}

int Ticket::getSum() const { return sum; }

void Ticket::setSum(int sum) { this->sum = sum; }

int Ticket::getId() const { return id; }

void Ticket::setId(int id) { this->id = id; }

ostream& operator<<(ostream& os, const Ticket& ticket)

{

os << "Сумма оплаты: " << ticket.sum << endl << "Номер: " << ticket.id << endl;

return os;

}

istream& operator>>(istream& is, Ticket& ticket) {

cout << "Введите ID номер: ";

ticket.id = InputError::Input(100000, 999999);

cout << "Введите сумму оплаты: ";

ticket.sum = InputError::Input(0, 1000000);

return is;

}

**Файл Trip.h:**

class Trip{

private:

string destantion;

int way;

int day;

int number;

public:

Trip(const string& destantion, int way = 0, int day = 0, int number = 0);

const string& getDestantion() const;

void setDestantion(const string& destaniton);

int getWay() const;

void setWay(int way);

int getDay() const;

void setDay(int day);

int getNumber() const;

void setNumber(int number);

friend ostream& operator <<(ostream& os, const Trip& trip);

friend istream& operator >> (istream& is, Trip& trip);

};

**Файл Trip.cpp:**

Trip::Trip(const string& desttantion, int way, int data, int number) :destantion(destantion), way(way), day(day), number(number) {}

const string& Trip::getDestantion() const { return destantion; }

void Trip::setDestantion(const string &destantion) { this->destantion = destantion; }

int Trip::getWay() const { return way; }

void Trip::setWay(int way) { this->way = way; }

int Trip::getDay() const { return day; }

void Trip::setDay(int day) { this->day = day; }

int Trip::getNumber() const { return number; }

void Trip::setNumber(int number) { this->number = number; }

ostream& operator << (ostream& os, const Trip& trip)

{

os << setw(20) << trip.destantion << "|" << setw(24) << trip.way << "|" << setw(4) << trip.day;

return os;

}

istream& operator >> (istream& is, Trip& trip){

cout << "Введите пункт назначения: ";

rewind(stdin);

getline(is, trip.destantion);

cout << "Введите далность пути (км): ";

trip.way = InputError::Input(0, 10000);

cout << "Введите номер дня: ";

trip.day = InputError::Input(1, 31);

trip.number = abs(2 \* trip.way - trip.day \* trip.day);

return is;

}

**Файл User.h:**

class User{

private:

string username;

string password;

bool admin;

public:

explicit User(string username = "", string password = "", bool admin = false);

const string& getUsername() const;

void setUsername(const string& username);

const string& getPassword() const;

void setPassword(const string& password);

bool isAdmin() const;

void setAdmin(bool admin);

User& operator=(const User& user);

friend ostream& operator << (ostream& os, const User& user);

friend istream& operator >> (istream& is, User& user);

};

**Файл User.cpp:**

User::User(string username, string password, bool admin) :username(username), password(password), admin(admin) { }

const string& User::getUsername() const { return username; }

void User::setUsername(const string& username) { this->username = username; }

const string& User::getPassword() const { return password; }

void User::setPassword(const string& password) { this->password = password; }

bool User::isAdmin() const { return admin; }

istream& operator >> (istream& is, User& user)

{

rewind(stdin);

cout << "Введите имя пользователя: ";

is >> user.username;

cout << "Введите пароль: ";

char BUFFER[20], c;

int i = 0;

while ((c = \_getch()) != '\r'){

\_putch('\*');

BUFFER[i] = c;

i++;

}

BUFFER[i] = '\0';

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

{

BUFFER[j] = KEY;

}

user.password = string(BUFFER);

user.admin = false;

return is;

}

User& User::operator=(const User& user)

{

this->username = user.username;

this->password = user.password;

this->admin = user.admin;

return \*this;

}

# **ПРИЛОЖЕНИЕ Д**

*(обязательное)*

Ведомость документов