Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

ОТЧЕТ

ПО УЧЕБНОЙ ПРАКТИКЕ

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Выполнил  студент гр. 451001  В. А. Пузик | |
|  |  | Проверил  асс. Е.Е. Фадеева | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Минск, 2025

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc199713666)

[1. КОМПАНИЯ ЗАО «МЕЖДУНАРОДНЫЙ ДЕЛОВОЙ АЛЬЯНС» 3](#_Toc199713667)

[2. ИНИВИДУАЛЬНОЕ ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ 5](#_Toc199713668)

[2.1. Техническое задание 5](#_Toc199713669)

[3. ВЫПОЛНЕНИЕ 13](#_Toc199713670)

[3.1 Разработка схемы программы 13](#_Toc199713671)

[3.2. Выделение основных структур данных 29](#_Toc199713672)

[3.3. Описание структур в виде таблиц 30](#_Toc199713673)

[3.4. Определение подпрограмм и их описание 32](#_Toc199713674)

[3.5 Тестирование и отладка программы 39](#_Toc199713675)

[4. РУКОВОДСТВО ПО УСТАНОВКЕ И ЭКСПЛУАТАЦИИ 43](#_Toc199713676)

[4.1. Установка программы 43](#_Toc199713677)

[4.2. Эксплуатация программы 43](#_Toc199713678)

[4.2.1. Запуск программы 43](#_Toc199713679)

[4.2.2. Основные функции программы 43](#_Toc199713680)

[4.2.3. Форматы ввода данных 44](#_Toc199713681)

[4.2.4. Работа с файлами 44](#_Toc199713682)

[4.3. Возможные проблемы и их решения 45](#_Toc199713683)

[Приложение 1. Код программы 46](#_Toc199713684)

# 1. КОМПАНИЯ ЗАО «МЕЖДУНАРОДНЫЙ ДЕЛОВОЙ АЛЬЯНС»

На днях наша группа побывала в ЗАО «Международный деловой альянс», где мы ознакомились с деятельностью их IT-департамента. Данная компания на протяжении многих лет удерживает лидерские позиции на рынке в сфере цифровых технологий и систем автоматизации для разнообразных отраслей. Созданная в начале 2000-х, она выросла из небольшой команды энтузиастов в крупного игрока, который воплощает в жизнь масштабные проекты по всей стране. Её портфолио насчитывает десятки крупных проектов, выполненных для государственных органов, промышленных и транспортных компаний. Компания заслужила репутацию надёжного партнёра, способного претворять в жизнь как масштабные, так и технологически сложные решения.

Следует особо подчеркнуть, что в последние годы «Международный деловой альянс» активно занимается разработкой белорусских аналогов зарубежных IT-продуктов, которые сейчас не представлены на местном рынке. Такой подход даёт возможность заказчикам избежать зависимости от иностранных технологий и продолжать внедрять передовые цифровые решения даже в условиях существующих ограничений.

Из всех реализованных компанией проектов моё внимание особенно привлекли решения в области интернета вещей (IoT). Примером такого проекта является r-Tube — IoT-система, предназначенная для непрерывного контроля за состоянием изоляции трубопроводов в теплосетях. Автономные датчики r-Tube дают возможность исключить человеческий фактор в процессе мониторинга предизолированных труб, вовремя выявлять предпосылки к авариям и предупреждать потенциальные прорывы.

Высокий уровень профессионализма команды и её инновационный подход к разработкам многократно получали признание на отраслевых конкурсах и форумах. Компания владеет рядом престижных наград, полученных за внедрение передовых цифровых решений и за вклад в развитие белорусской IT-индустрии.

Нам также подробно рассказали о структуре компании. В составе «Международного делового альянса» функционируют несколько основных подразделений: департаменты разработки ПО, тестирования, внедрения и последующего сопровождения, а также аналитический центр и служба техподдержки. Подобный подход даёт возможность не просто создавать новые продукты, но и гарантировать их качественную интеграцию и последующее развитие. В компании работают специалисты самых разных направлений — от программистов и аналитиков до инженеров и менеджеров проектов, что позволяет формировать сильный и профессиональный коллектив.

Кроме того, в памяти остался один из проектов, который компания реализовала для Альфа-Банка. Это передовая система, позволяющая существенно ускорить обработку клиентских заявок и повысить удобство и надёжность обслуживания. За счёт внедрённого решения Альфа-Банк получил возможность оперативно отвечать на запросы клиентов, сводить к минимуму число ошибок и улучшать общий уровень сервиса. Было познавательно увидеть, как передовые IT-инструменты могут преобразовывать стандартные процессы, делая их более простыми и доступными для всех пользователей.

Во время визита нам поведали о том, какое значение придаётся поддержке молодых специалистов. В компании действуют активные программы менторства, систематически организуются хакатоны и внутренние соревнования, на которых студенты и начинающие специалисты могут испытать себя в решении практических задач. Наиболее удачные идеи имеют шанс на воплощение в коммерческих проектах. К тому же, молодым сотрудникам открыт доступ к передовым технологиям и предоставлена возможность работать под началом опытных менторов, что содействует их профессиональному росту.

Не меньшее впечатление на меня произвела и инфраструктура компании: светлые и просторные офисы, технологичные рабочие места, отлично оснащённые серверные помещения и комфортные зоны отдыха. Такое внимание к деталям и забота о комфорте персонала формируют атмосферу, благоприятную для работы и профессионального развития. Очевидно, что компания старается идти в ногу со временем, уделяя значительное внимание не только технологической составляющей, но и корпоративной культуре.

Эта экскурсия стала для меня крайне полезным опытом. Я смог увидеть изнутри, как организована работа в одной из ведущих IT-компаний, и осознал, насколько велика роль командной слаженности, непрерывного обучения и обмена опытом. Особенно интересно было ощутить, что здесь по-настоящему ценят инициативность и поддерживают свежие идеи, тем самым открывая перед молодыми специалистами реальные перспективы для роста. Теперь я ещё больше мотивирован развиваться в области информационных технологий и стремиться стать частью подобной профессиональной среды, где каждый сотрудник имеет возможность внести личный вклад в разработку передовых цифровых решений.

# 2. ИНИВИДУАЛЬНОЕ ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

### 2.1. Техническое задание

|  |  |
| --- | --- |
| **УТВЕРЖДАЮ**  Должность  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ФИО  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г. |  |
| Программное средство  наименование вида ИС  Система подбора комплектующих для ПК  Сокращенное наименование ИС  Техническое задание  Действует с \_\_\_\_10.03.25\_\_\_\_ | |
| **СОГЛАСОВАНО**  Должность  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ФИО  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025\_ г. | **РАЗРАБОТЧИК**  Должность  Пузик В. А.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г. |
| Минск 2025 | |

1. **ОБЩИЕ СВЕДЕНИЯ**

Полное наименование Системы и её условное обозначение

Система подбора комплектующих для ПК.

**1.4. Плановые сроки начала и окончания работ**

Начало: 10.03.25. Окончание: 31.05.25

* 1. **Назначение Системы**

Программное средство предназначено для автоматизации подбора комплектующих для ПК, формирования совместимых конфигураций в заданном ценовом диапазоне и последующего оформления заказа. Система обеспечивает полный цикл работы с данными о комплектующих, их типах и совместимости, включая добавление, редактирование, удаление и поиск информации.

**2.2. Цели создания Системы**

* Управление данными о комплектующих:
  + Создание и поддержание в актуальном состоянии базы данных комплектующих, их типов и списков совместимости.
  + Обеспечение возможности добавления, редактирования и удаления записей о комплектующих.
* Подбор конфигураций и оформление заказа:
  + Автоматизация подбора всех возможных вариантов комплектации ПК в заданном ценовом диапазоне с сортировкой по цене.
  + Предоставление функционала для оформления заказа на выбранный вариант конфигурации.
* Поиск и анализ информации:
  + Реализация функции поиска комплектующих определённого типа, совместимых с конкретным устройством.
  + Обеспечение возможности просмотра всех списков для удобного анализа данных.

**3. ХАРАКТЕРИСТИКАОБЪЕКТА АВТОМАТИЗАЦИИ**

Объектом автоматизации является деятельность по управлению списком комплектующих, подбору конфигураций ПК из совместимых комплектующих, формированию вариантов сборок в заданном ценовом диапазоне и обработке заказов.

**3.1. Структура объекта автоматизации**

Деятельность по подбору комплектующих для ПК включает следующие основные процессы:

* Управление каталогом комплектующих:
  + Учет компонентов с их характеристиками (код, фирма, модель, цена, наличие).
  + Ведение нормативно-справочной информации по типам комплектующих и их совместимости.
* Подбор и формирование конфигураций:
  + Поиск совместимых компонентов для построения полной сборки.
  + Формирование и расчет стоимости различных вариантов комплектации в заданном ценовом диапазоне.
* Работа с заказами:
  + Прием и обработка заказов от пользователей на выбранные конфигурации.
  + Фиксация окончательной сборки и ее стоимости.

**3.2. Особенности объекта автоматизации**

* Сложная структура совместимости:
  + Тесная связь между комплектующими, их типами и правилами совместимости.
  + Необходимость поддержания целостности данных для формирования работоспособных сборок.
* Ориентация на пользователя:
  + Приоритет подбора конфигурации в заданном пользователем ценовом диапазоне.
  + Необходимость предоставления всех допустимых вариантов сборки для обеспечения выбора.
* Требования к точности данных:
  + Критичность правильности информации о совместимости компонентов.
  + Важность актуальности цен и данных о наличии для корректного формирования итогового заказа.

**4. ТРЕБОВАНИЯ К СИСТЕМЕ**

**4.1.1.1. Требования к структуре и функционированию системы**

1. **Подсистема управления каталогом:** Обеспечивает ввод и добавление в систему новых комплектующих, их типов и правил совместимости с проверкой корректности вводимых данных.
2. **Подсистема подбора конфигураций:** Обеспечивает подбор всех возможных вариантов сборки компьютера из совместимых комплектующих в рамках заданного пользователем ценового диапазона.
3. **Подсистема оформления заказа:** Управляет процессом формирования заказа на выбранный пользователем вариант конфигурации ПК.
4. **Подсистема проверки ввода:** Обеспечивает проверку на корректность и достоверность вводимых данных для всех типов объектов системы (коды, цены и т.д.).
5. **Подсистема считывания файлов:** Загружает данные о комплектующих, их типах и списках совместимости из внешних файлов в оперативную память системы.
6. **Подсистема сохранения данных:** Обеспечивает сохранение всех данных системы (каталоги, заказы) в типизированные файлы для последующего использования.
7. **Подсистема удаления элементов:** Обеспечивает корректное удаление записей из списков комплектующих, типов и правил совместимости с проверкой зависимостей.
8. **Подсистема редактирования:** Позволяет изменять характеристики существующих комплектующих (цену, модель, параметры) с учетом ограничений целостности данных.
9. **Подсистема поиска комплектующих:** Выполняет поиск комплектующих заданного типа, которые совместимы с конкретным, указанным пользователем, компонентом.
10. **Подсистема сортировки:** Организует сортировку списка подобранных конфигураций ПК по возрастанию цены для удобства выбора.
11. **Подсистема формирования файла заказа:** Создает текстовый файл с детальной информацией о финальной сборке и ее стоимости.
12. **Подсистема управления пользовательским интерфейсом:** Обеспечивает взаимодействие пользователя с системой через систему меню и подменю в консольном режиме.

**3.4. Описание подсистем программы**

Таблица 3.4. – Описание подсистем программы

|  |  |  |  |
| --- | --- | --- | --- |
| **Подсистема** | **Задача** | **Функции** | **Требования** |
| Управление каталогом | Обеспечивает ввод и добавление новых комплектующих, их типов и правил совместимости в систему | Обработка пользовательского ввода с проверкой корректности данных. Добавление новых записей в соответствующие списки | Корректность добавления новых элементов. Проверка уникальности кодов. Оповещение пользователя о результате операции |
| Подбор конфигураций | Подбор конфигураций | Получение от пользователя ценового диапазона. Перебор комбинаций комплектующих с проверкой совместимости | Корректный учет всех правил совместимости. Точность расчета итоговой цены. Предоставление всех допустимых вариантов |
| Оформление заказа | Управляет процессом заказа выбранной пользователем конфигурации | Выбор конфигурации из списка. Формирование итогового заказа. Создание информационного файла с деталями сборки | Проверка, что выбранный вариант существует. Корректная фиксация всех комплектующих в заказе. Генерация файла с полным описанием заказа |
| Считывание файлов | Загружает данные из внешних файлов в оперативную память | Чтение данных о комплектующих, их типах и совместимости из типизированных файлов. Преобразование данных в структуры списков | Корректное восстановление всех данных и связей между ними. Обработка ситуаций с отсутствующими файлами |

Таблица 3.4. – Продолжение

|  |  |  |  |
| --- | --- | --- | --- |
| **Подсистема** | **Задача** | **Функции** | **Требования** |
| Удаление элементов | Обеспечивает корректное удаление записей из списков | Определение элемента для удаления. Проверка на наличие зависимостей в других списках (например, в списке совместимости). Обновление связей | Сохранение целостности данных. Проверка зависимостей перед операцией удаления |
| Редактирование | Позволяет изменять характеристики существующих объектов | Выбор объекта для редактирования. Изменение выбранных параметров (цена, модель, фирма). Сохранение изменений | Проверка корректности новых значений (например, цена не может быть отрицательной). |
| Поиск совместимых комплектующих | Выполняет поиск комплектующих заданного типа, совместимых с конкретным компонентом | Ввод кода основного компонента и типа искомых. Фильтрация списка комплектующих на основе правил совместимости | Корректность алгоритма поиска по списку совместимости. Отображение всех найденных результатов |
| Сортировка | Организует сортировку списка подобранных конфигураций | Сортировка итогового списка сборок ПК по возрастанию цены | Корректность алгоритма сортировки. |
| Формирование файла заказа | Создает текстовый файл с детальной информацией о выбранной конфигурации | Сбор данных о каждом комплектующем в заказе. Запись информации в текстовый файл с указанием общей стоимости | Корректное формирование файла. Включение всей необходимой информации |
| Управление пользовательским интерфейсом | Обеспечивает взаимодействие пользователя с системой | Отображение меню и подменю. Обработка пользовательского ввода. Навигация между разделами | Интуитивно понятный интерфейс. Информативные сообщения о результатах операций |

**4.2 Функциональные требования**

Программа должна содержать пользовательское меню, состоящее из следующих пунктов:

1. Чтение данных из файла
   1. Загрузка данных о типах комплектующих, комплектующих, совместимости и заказах из соответствующих файлов
   2. Запрос подтверждения у пользователя перед заменой текущих данных в памяти
   3. Вывод сообщений об успешной загрузке или ошибках для каждого типа данных
2. Просмотр всего списка
   1. Подменю с выбором списка для просмотра:
      1. Типы комплектующих
      2. Комплектующие
      3. Совместимость комплектующих
      4. Заказы
   2. Постраничный вывод данных при большом объеме информации
3. Сортировка данных
   1. Выбор типа данных для сортировки:
      1. Типы комплектующих (по коду)
      2. Комплектующие (по цене, коду или производителю)
      3. Заказы (по дате)
      4. Сортировка выбранного списка с соответствующим критерием
4. Поиск данных с использованием фильтров
   1. Поиск комплектующих по различным критериям:
      1. По диапазону цен (минимальная и максимальная цена)
      2. По производителю
      3. По типу комплектующих
      4. По модели
      5. По минимальному количеству в наличии
   2. Отображение результатов поиска в виде списка
5. Добавление данных в список
   1. Подменю с выбором типа данных для добавления:
      1. Типы комплектующих
      2. Комплектующие
      3. Правила совместимости
      4. Пошаговый ввод данных с проверкой корректности
   2. Автоматическое присвоение уникального ID
6. Удаление данных из списка
   1. Подменю с выбором типа данных для удаления
   2. Выбор конкретной записи для удаления
   3. Подтверждение операции удаления
   4. Проверка зависимостей перед удалением
7. Редактирование данных
   1. Подменю с выбором типа данных для редактирования
   2. Выбор конкретной записи и параметра для изменения
   3. Проверка корректности новых значений
8. Сборка ПК и Заказы
   1. Подменю специальных функций, включающее:
      1. Подбор вариантов комплектации ПК в заданном ценовом диапазоне
      2. Ввод минимальной и максимальной цены
      3. Автоматический подбор всех возможных сборок из совместимых комплектующих
      4. Сортировка найденных вариантов по возрастанию итоговой стоимости
      5. Сохранение результатов в текстовый файл
   2. Оформление заказа
      1. Выбор варианта комплектации из ранее подобранных
      2. Ввод данных заказчика (имя, телефон)
      3. Проверка наличия комплектующих на складе
      4. Автоматическое уменьшение количества товара в наличии
      5. Формирование детального файла заказа с составом сборки и итоговой ценой
   3. Поиск совместимых комплектующих заданного типа
      1. Выбор базовой комплектующей из списка
      2. Выбор типа комплектующих для поиска совместимых
      3. Поиск и отображение всех совместимых комплектующих указанного типа
9. Выход из программы без сохранения изменений
   1. Немедленное завершение работы программы без сохранения изменений
   2. Освобождение выделенной памяти
10. Выход с сохранением изменений
    1. Сохранение всех списков (типы комплектующих, комплектующие, совместимость, заказы) в соответствующие файлы
    2. Подтверждение успешного сохранения
    3. Корректное завершение работы программы с освобождением памяти

**4.3. Обработка ошибок**

* Ошибки ввода данных:
  + Ввод некорректного значения (ожидался ввод числа, а получена строка и наоборот; пустая строка или значение за пределами допустимого диапазона).
  + Ввод отрицательных значений для цены, количества на складе или кодов элементов.
  + Превышение максимальной длины текстовых полей (производитель, модель, название типа).
* Ошибки добавления данных:
  + Добавление комплектующей для несуществующего типа.
  + Создание правила совместимости между несуществующими комплектующими.
  + Попытка добавления записи с уже существующим уникальным идентификатором.
  + Добавление пустых или некорректно заполненных записей.
* Ошибки редактирования:
  + Попытка изменения несуществующего элемента.
  + Ввод недопустимых значений при редактировании (например, отрицательная цена или некорректный код типа).
  + Изменение кода элемента на уже существующий в системе.
* Ошибки удаления:
  + Попытка удаления несуществующего элемента.
  + Удаление типа комплектующих, для которого существуют зависимые комплектующие.
  + Удаление комплектующей, которая используется в правилах совместимости или заказах.
* Ошибки поиска и сортировки:
  + Выполнение операций поиска или сортировки при пустых списках.
  + Использование некорректных параметров фильтрации (например, максимальная цена меньше минимальной).
  + Поиск по несуществующим критериям или значениям.
* Ошибки работы с файлами:
  + Отсутствие файлов данных для загрузки.
  + Ошибки при чтении или записи данных в файл (например, поврежденный файл или отсутствие прав доступа).
  + Некорректная структура данных в загружаемых файлах.
* Ошибки подбора конфигурации ПК:
  + Ввод некорректного ценового диапазона (максимальная цена меньше минимальной).
  + Отсутствие совместимых комплектующих в заданном ценовом диапазоне.
  + Выполнение подбора при пустых списках комплектующих или правил совместимости.
* Ошибки оформления заказов:
  + Попытка оформления заказа для несуществующего варианта комплектации.
  + Недостаточное количество товара на складе для комплектующих из выбранной конфигурации.
  + Ввод некорректных данных заказчика (пустые поля имени или телефона).
  + Попытка оформления заказа при отсутствии подобранных вариантов комплектации.
* Ошибки поиска совместимости:
  + Поиск совместимых комплектующих для несуществующей базовой комплектующей.
  + Указание несуществующего типа комплектующих для поиска.
  + Выполнение поиска при пустых списках комплектующих или правил совместимости.
* Ошибки вызовов функций:
  + Попытка выполнения операций (добавление, редактирование, удаление, сортировка, поиск) при неинициализированных или пустых списках.
  + Обращение к неинициализированным данным или указателям.
  + Попытка выполнения специальных функций без предварительной загрузки необходимых данных.

Для каждой ошибки система должна предоставлять пользователю понятное сообщение с указанием причины ошибки и рекомендацией по ее устранению. При критических ошибках работы с памятью или файлами программа должна корректно завершать работу с освобождением всех выделенных ресурсов.

# 3. ВЫПОЛНЕНИЕ

### 3.1 Разработка схемы программы

Данная схема иллюстрирует последовательность выполнения операций в основном меню программы, начиная с инициализации указателей на списки и счётчиков. На каждой итерации цикла пользователю предоставляется список возможных действий, после чего программа обрабатывает введённые данные, проверяет их корректность и выполняет соответствующее действие.

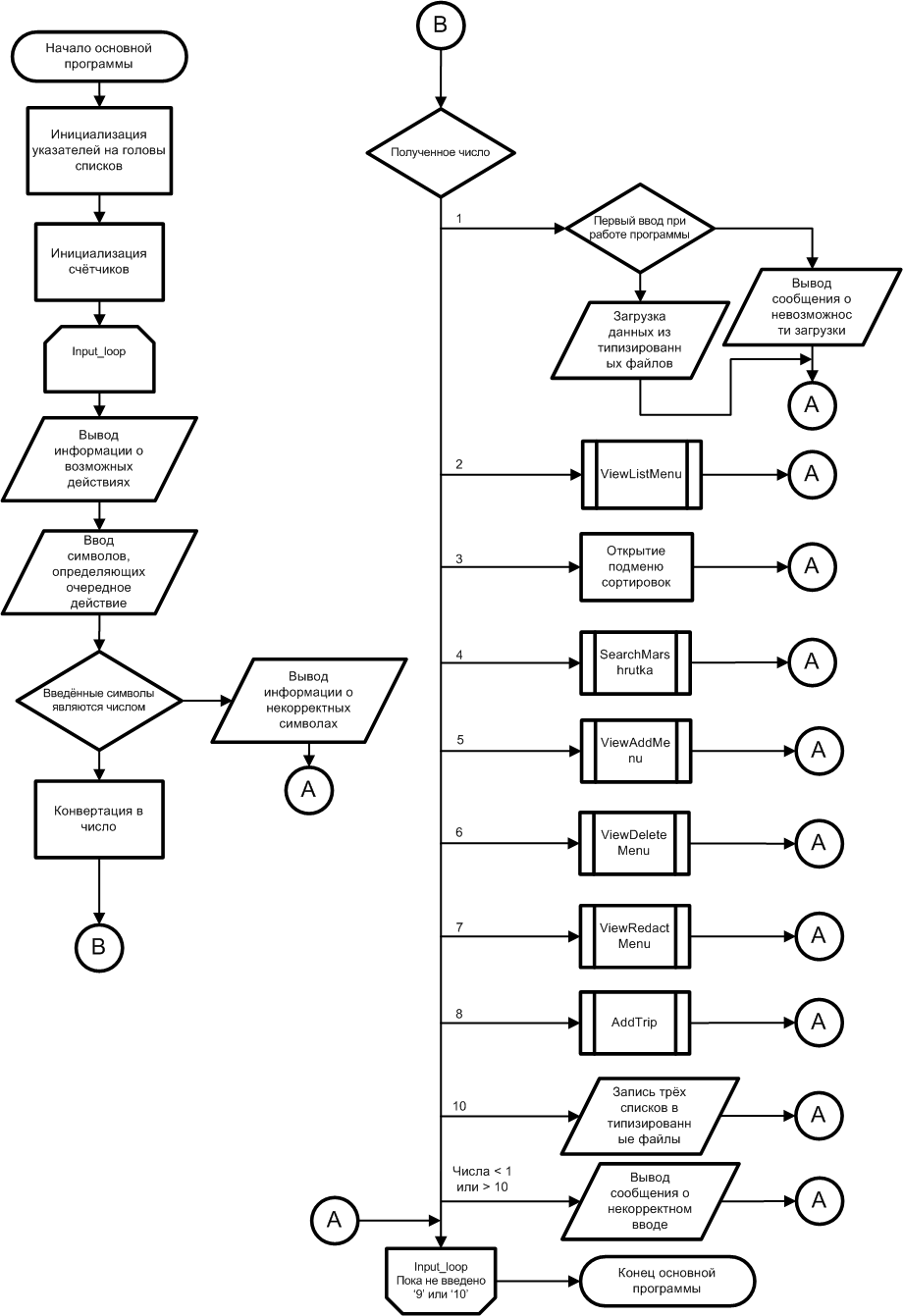


Рисунок 3.1 – Схема работы основного меню программы

Процедура проверки номера маршрутки: проверяются два допустимых в РБ формата, обычные автомобильные номера и жёлтые номера, для транспорта, связанного с перевозками пассажиров. Процедура изменения количества мест на выезде: проверяет, чтобы не превышалась общая вместимость маршрутки.

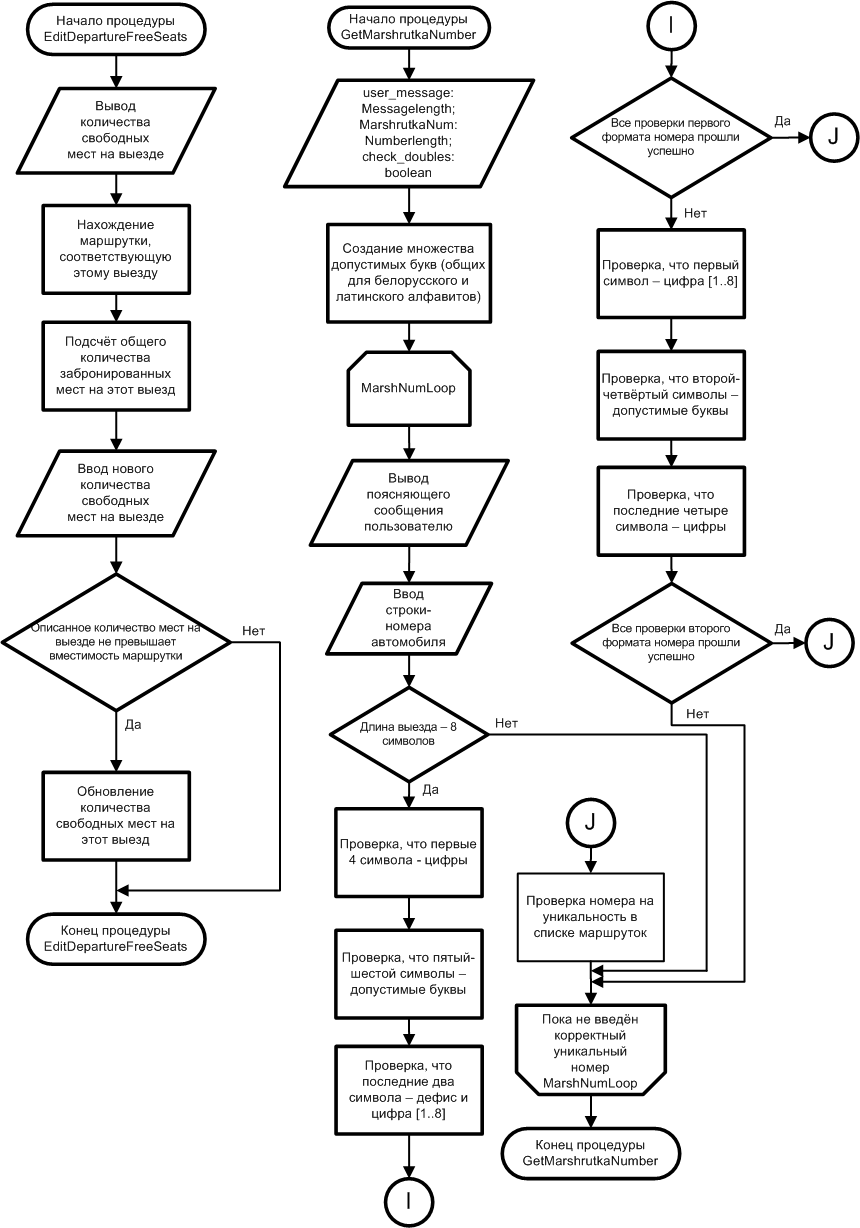


Рисунок 3.2 – Схема работы процедур проверки автомобильного номера и изменения количества мест на выезде

Процедура добавления маршрутки. Основные этапы процедуры включают ввод данных о маршрутке, их проверку и добавление нового элемента в список.



Рисунок 3.3 – Схема работы процедуры добавления маршрутки

Процедура проверки вводимых даты и времени на корректность. Используются как характеристики выезда и маршрутки соответственно. В первой процедуре проверяется формат "ММ.ДД", диапазоны месяца [1, 12] и дня [1, 31], а также соответствие количества дней в месяце. Во второй проверяется формат "ЧЧ:ММ", диапазоны часов [0, 23] и минут [0, 59]. При ошибке выводится сообщение, и ввод повторяется до получения корректного значения.

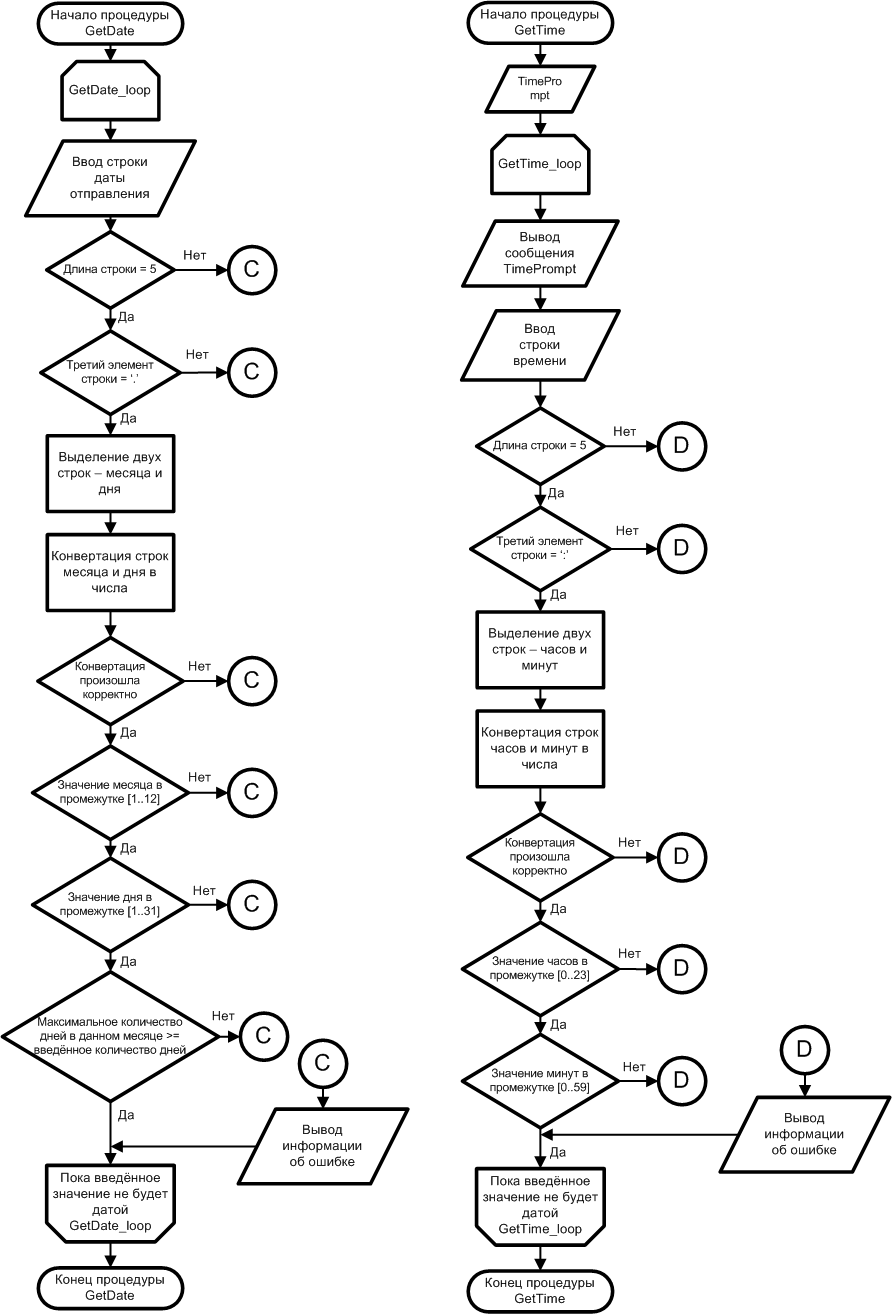


Рисунок 3.4. – Схема работы процедур проверок корректного ввода даты и времени

Процедура предназначена для проверки корректности ввода стоимости проезда. Сначала пользователю выводится сообщение с инструкцией, после чего он вводит строку стоимости. Далее проверяется наличие десятичной точки, правильность её позиции и количество символов после неё (не более двух). После успешной проверки строка конвертируется в положительное число, завершая выполнение процедуры.

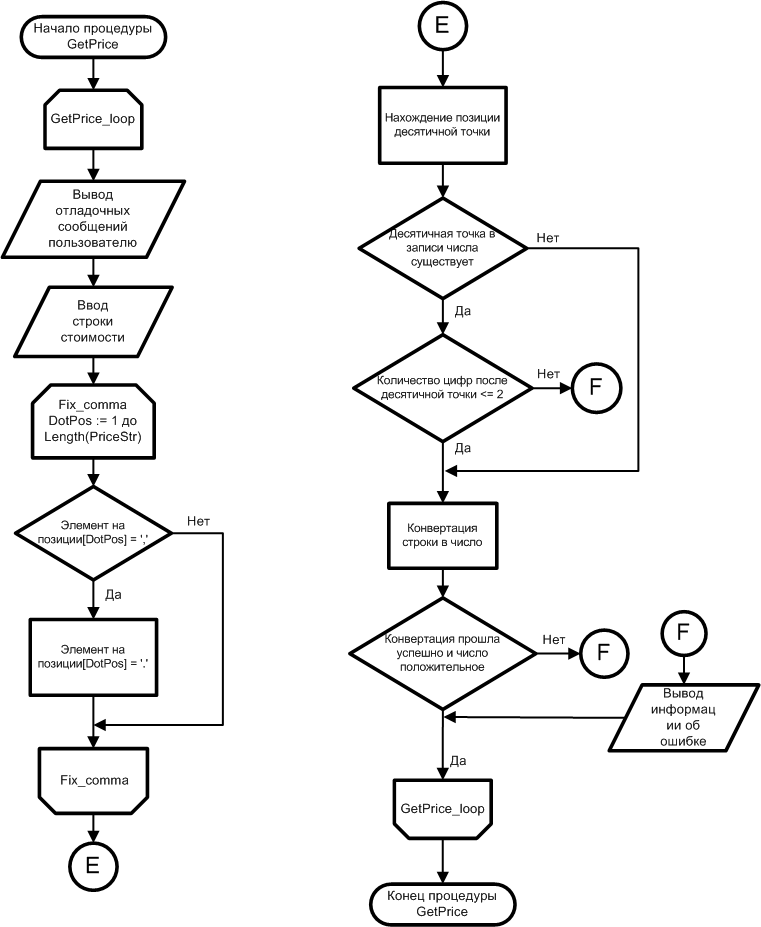


Рисунок 3.5. – Схема работы процедуры проверки корректного ввода стоимости

Процедуры ViewMarshrutkaList и DeleteMarshrutka предназначены для работы со списком маршруток. Первая выводит список маршруток, проверяя его заполненность, и отображает общее количество элементов. Вторая удаляет маршрутку по заданному номеру, проверяя наличие выездов и корректно освобождая память.

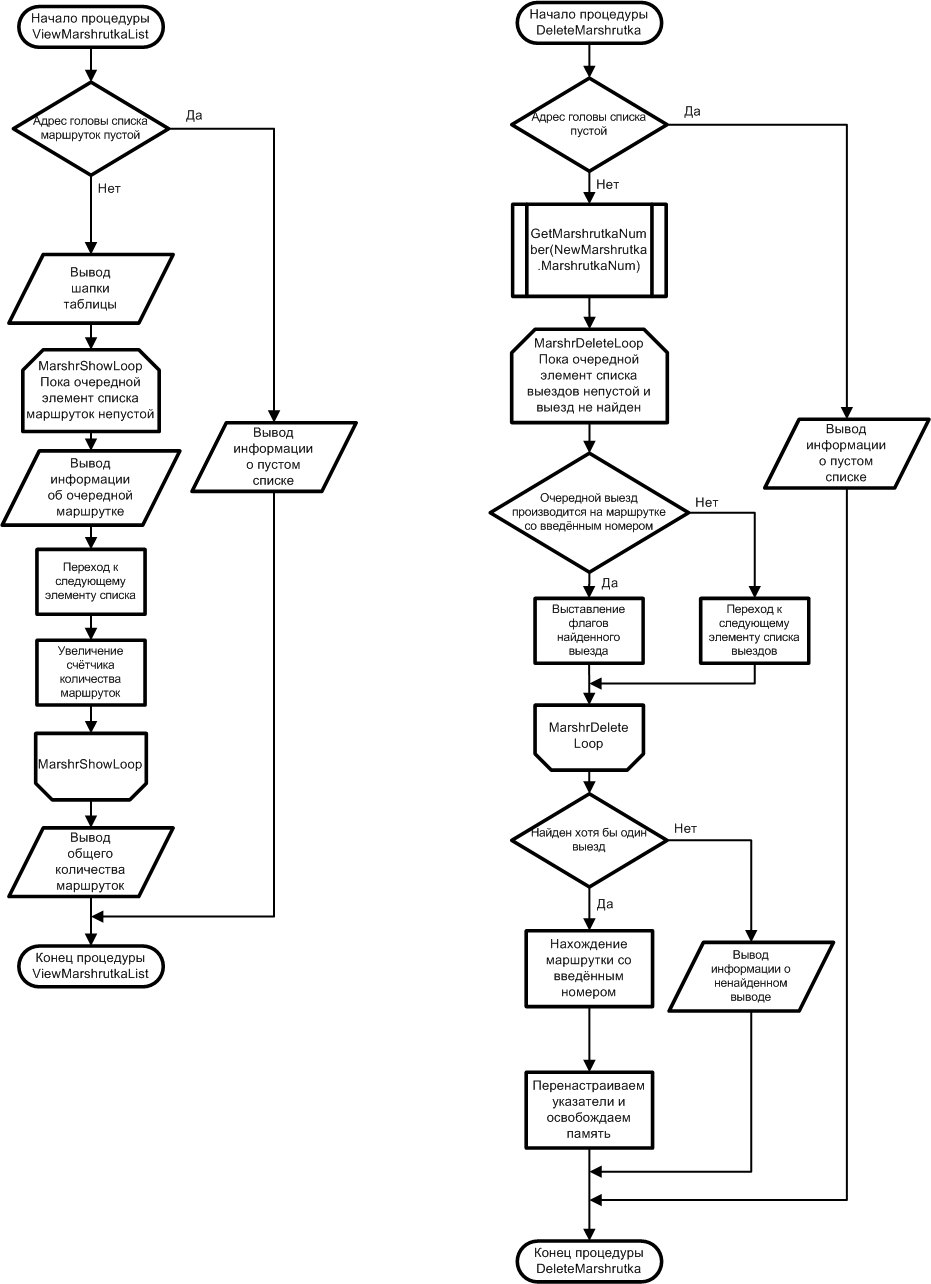


Рисунок 3.6. – Схема работы процедуры проверки корректного ввода стоимости

Процедуры реализуют проверку возможности изменения элемента списка маршруток и само изменение. Первая процедура проверяет существование выездов для маршрутки, ввод новых характеристик и их корректность, после чего обновляет данные. Вторая предоставляет меню редактирования, организуя ввод номера маршрутки и переход к процедуре изменения.

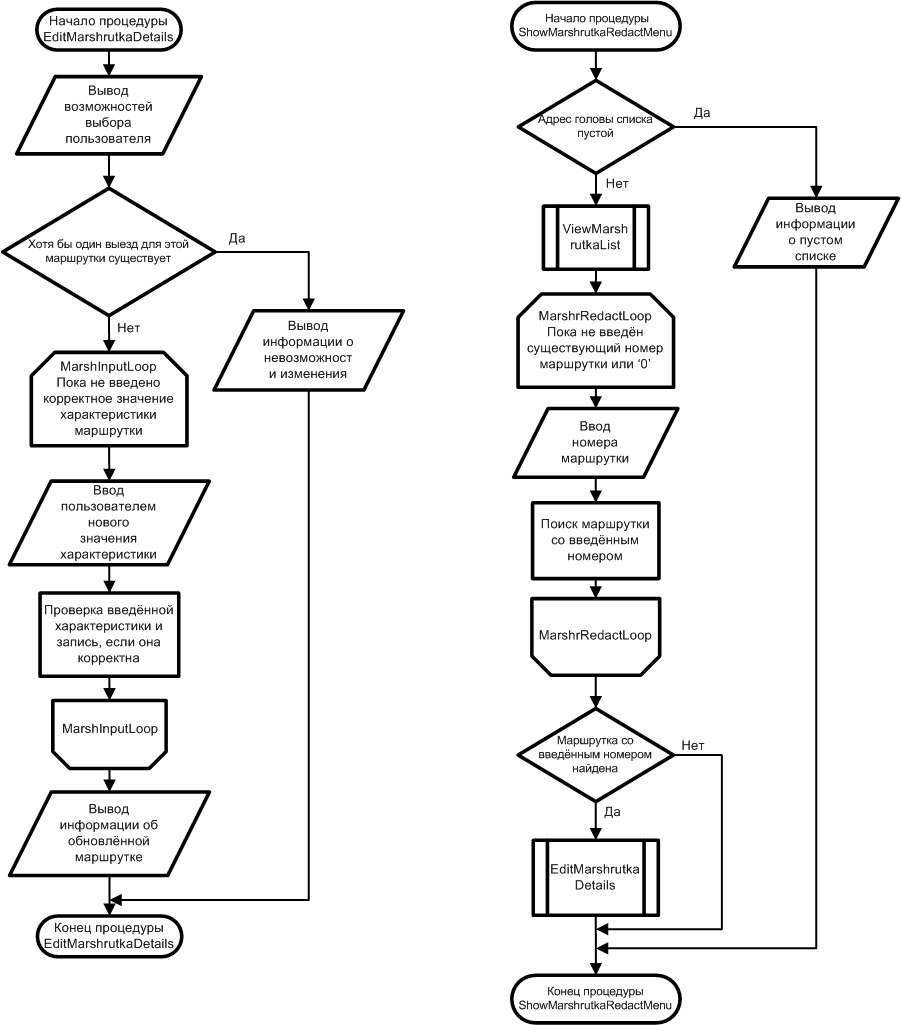


Рисунок 3.7. – Схема работы процедуры изменения элемента списка маршруток

Процедуры описывают добавление нового выезда в список и просмотр существующих выездов. Первая процедура проверяет существование маршрутки, запрашивает дату выезда, создаёт новый элемент списка и добавляет его в конец. Вторая процедура выводит информацию о всех выездах, проверяя, пуст ли список, и организует их последовательный просмотр.

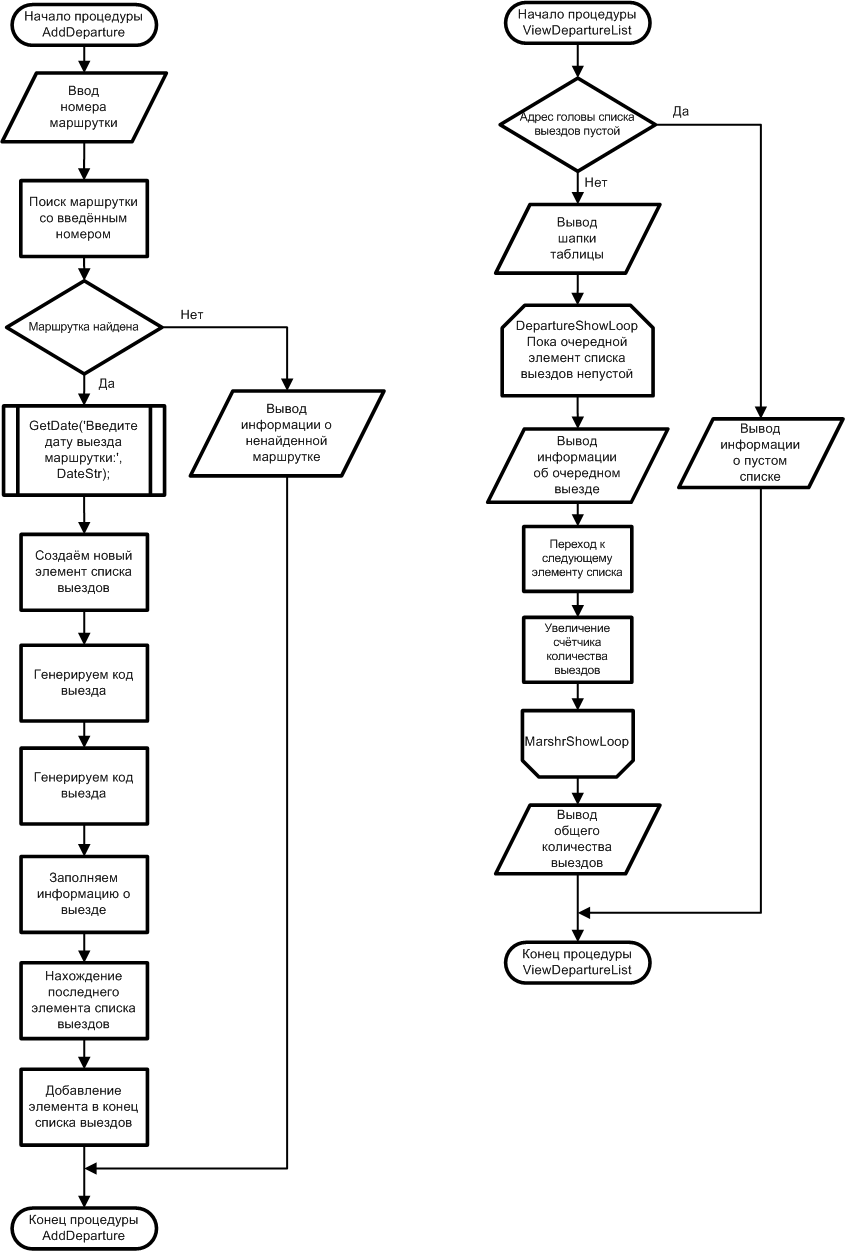


Рисунок 3.8. – Схема работы процедуры вывода списка выездов и добавления в него элементов

Процедура удаления элемента списка выездов. Проверяется наличие выездов в списке, корректность введённого кода и активные бронирования. Если код введён верно и бронирований нет, процедура находит элемент, перенастраивает указатели и освобождает память. При невозможности удаления выводится соответствующее сообщение.

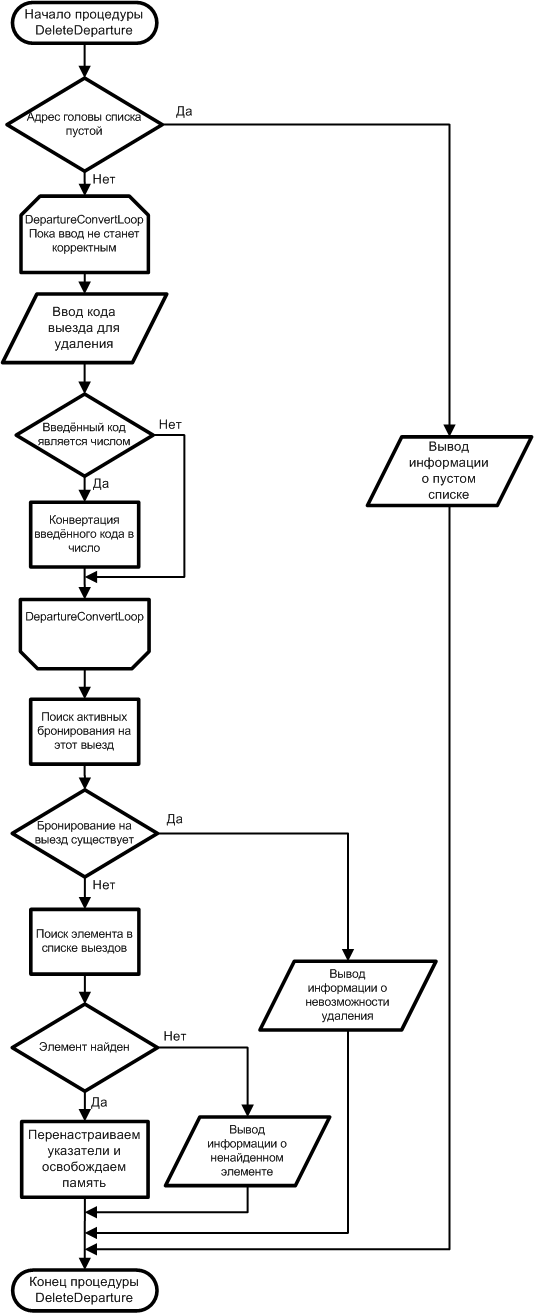


Рисунок 3.9. – Схема работы процедуры удаления элемента списка выездов.

Процедуры проверки возможности редактирования и изменения выезда. Сначала проверяется наличие элементов в списке, если он пуст, выводится уведомление. Пользователь вводит код выезда, который проверяется на существование. Изменение характеристик возможно только при отсутствии активных бронирований.

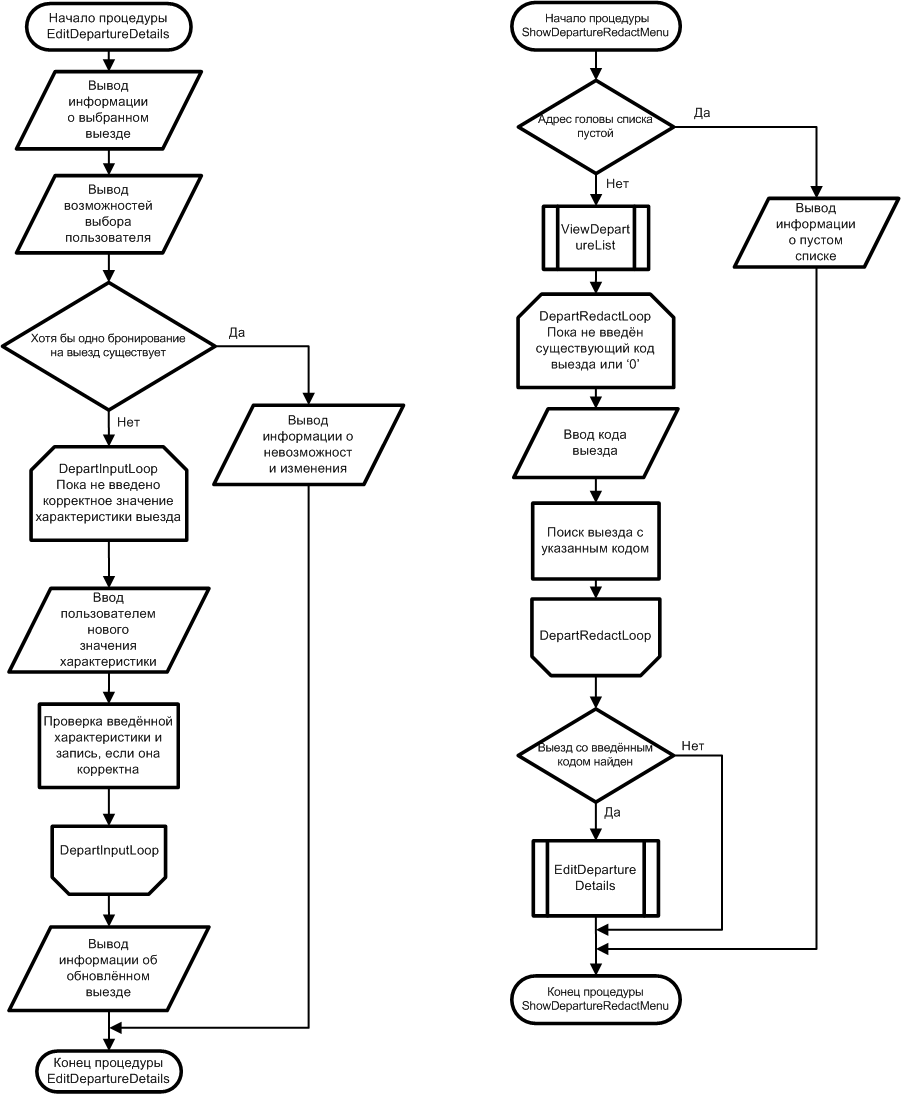


Рисунок 3.10. – Схема работы процедуры изменения элемента списка выездов

Схема показывает процесс добавления бронирования: вводится код выезда, проверяется его корректность и наличие. Затем вводятся характеристики бронирования, которые проверяются на правильность. После этого создаётся запись в списке бронирований, уменьшается количество свободных мест.

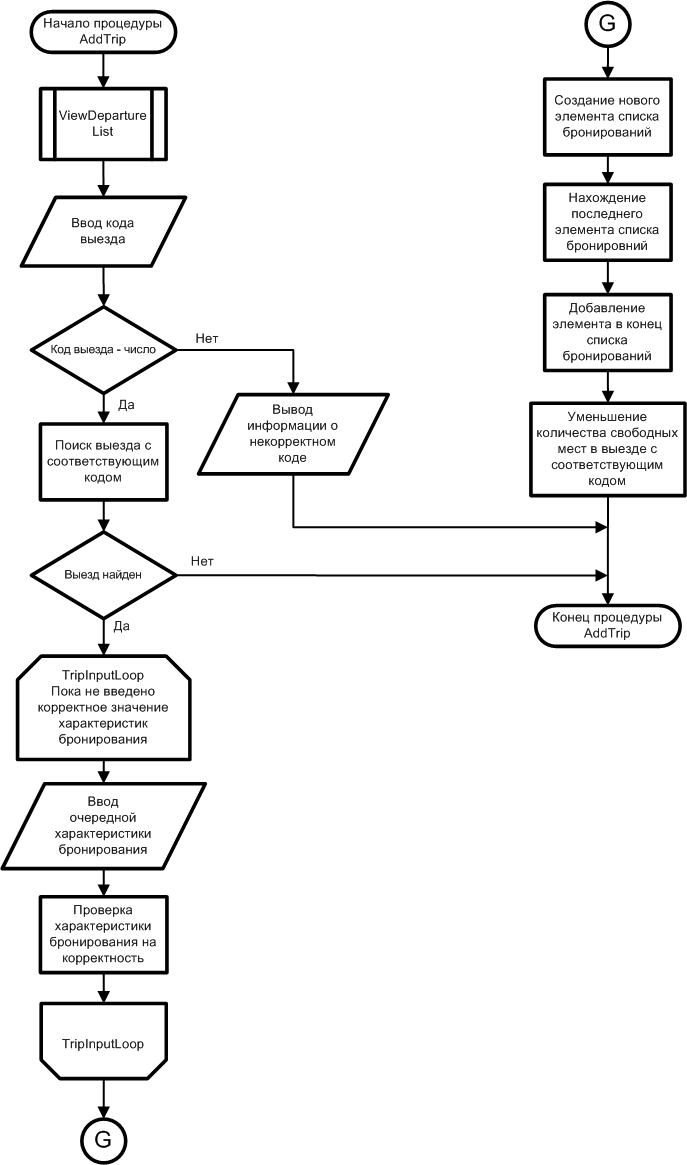


Рисунок 3.11. – Схема работы процедуры добавления бронирования

Первая процедура проверяет, есть ли бронирования. Если они есть, выводится таблица с деталями и общим количеством. Если нет, показывается сообщение. Вторая процедура проверяет наличие списка. При наличии запрашивается код для удаления, который проверяется и ищется в списке. Если элемент найден, он удаляется, иначе выводится сообщение. Если список пуст, также выводится сообщение.

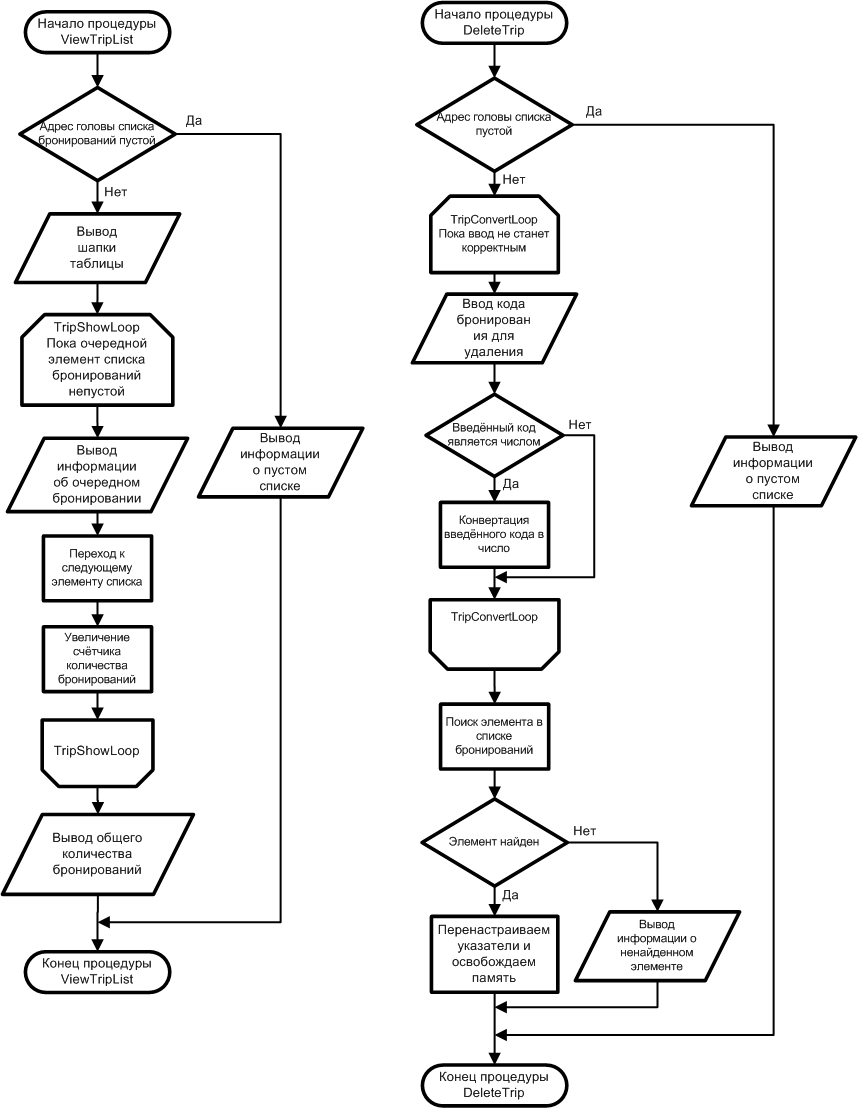


Рисунок 3.12. – Схема работы процедуры вывода и удаления элемента списка бронирований

Процедуры, отвечающие за проверку возможности изменения элемента списка бронирований и само изменение сначала показываются данные бронирования, затем пользователь выбирает параметр для изменения, вводит новое значение, которое проверяется и сохраняется. После этого отображается обновлённая информация.

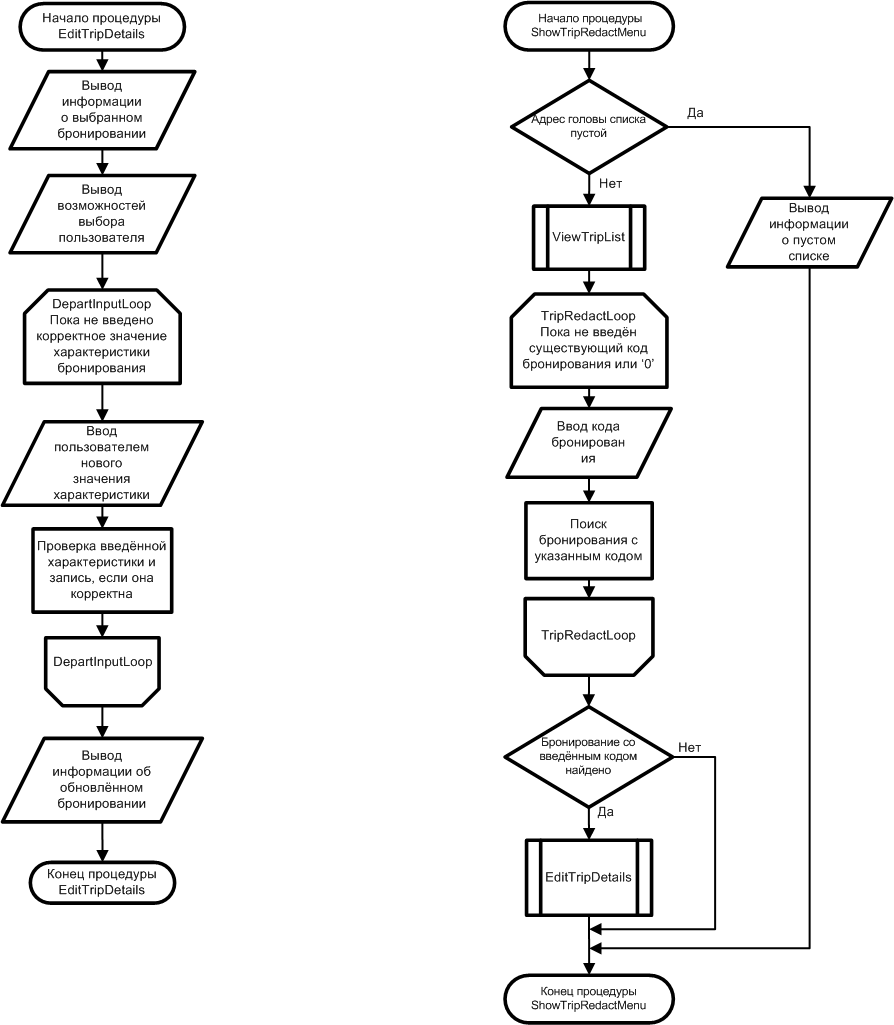


Рисунок 3.13. – Схема работы процедуры изменения элемента списка бронирований

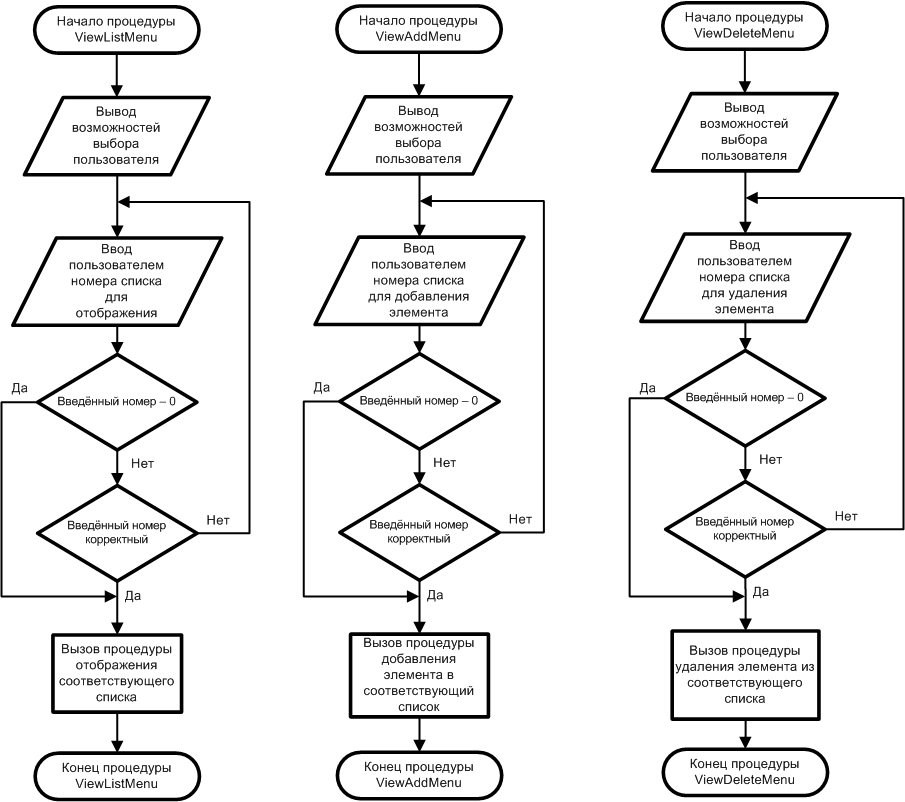


Рисунок 3.14. – Процедуры работы основных подменю программы

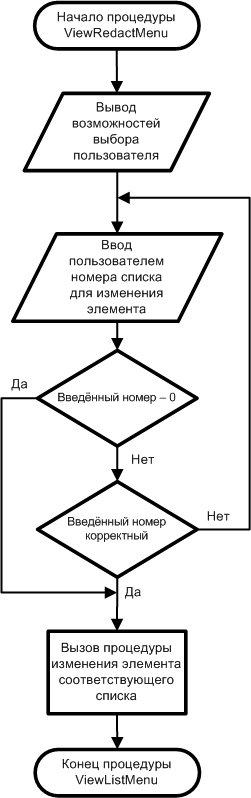


Рисунок 3.15. – Процедуры работы основных подменю программы (часть 2)

Процедура поиска маршрутки: вводятся пункты, даты и выводится шапка таблицы. В цикле проверяются маршрутки: совпадают ли пункты и даты. При совпадении выводится информация о выезде. Далее переход к следующей маршрутке.

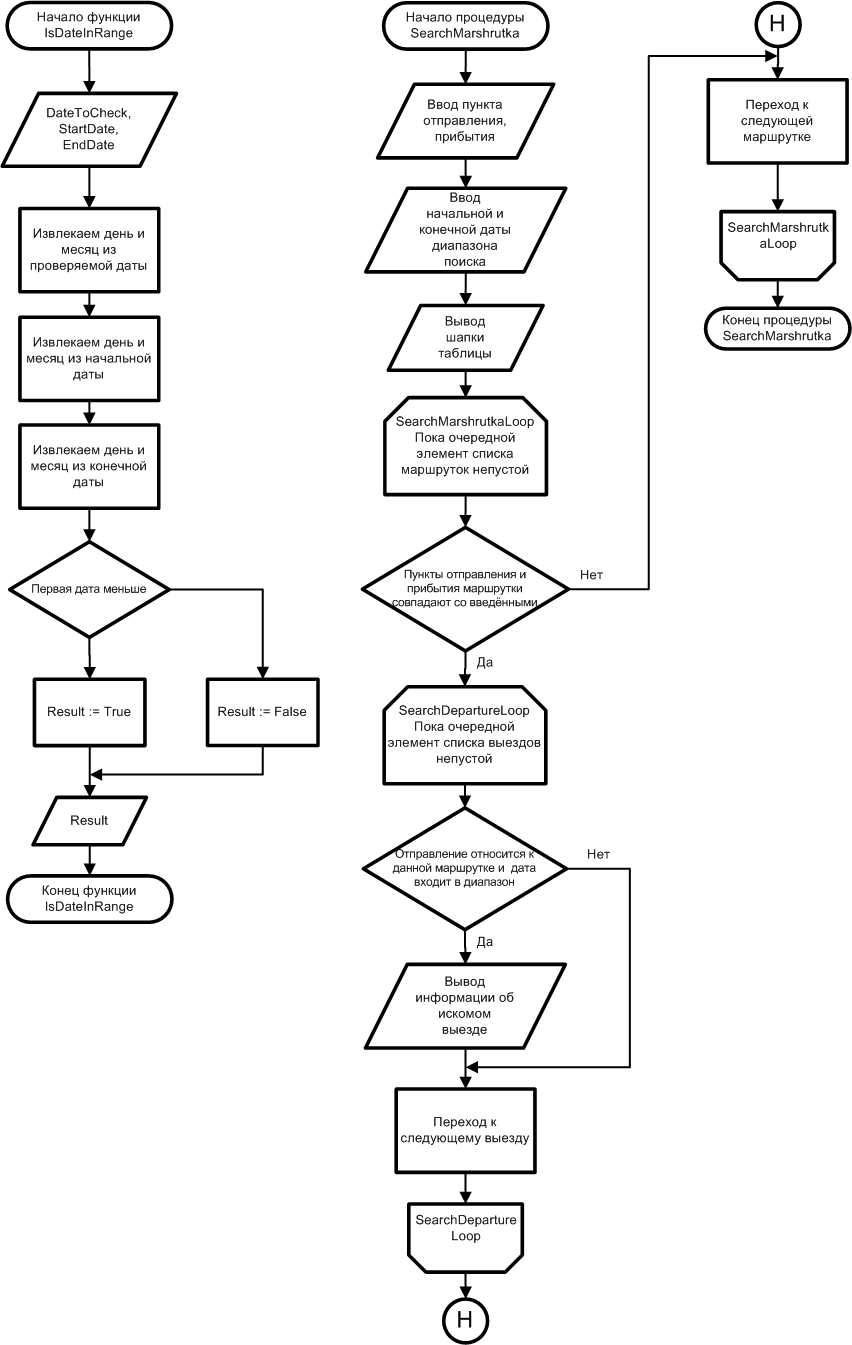


Рисунок 3.16. – Схема работы процедуры поиска маршрутки по введённым параметрам

### 3.2. Выделение основных структур данных

Таблица 3.1 используемые идентификаторы

|  |  |  |
| --- | --- | --- |
| Имя идентификатора структуры | Назначение структуры | Тип структуры |
| TFixedString | Строка фиксированной длины для использования в типизированных файлах | Массив символов |
| TComponentType | Хранение информации о типах комплектующих | Запись (record) |
| TComponent | Хранение информации о комплектующих | Указатель |
| T\_Departure | Хранение информации о выездах | Запись (record) |
| T\_List\_Departure | Элемент списка выездов | Запись (record) |
| T\_Pointer\_Departure | Указатель на элемент списка выездов | Указатель |
| T\_Trip | Хранение информации о бронированиях | Запись (record) |
| T\_List\_Trip | Элемент списка бронирований | Запись (record) |
| T\_Pointer\_Trip | Указатель на элемент списка бронирований | Указатель |
| T\_Status | Статус бронирования (подтверждено/не подтверждено) | Перечисляемый тип |
| Timelength | Строка для хранения времени | Строковый тип фиксированной длины |
| Numberlength | Строка для хранения номеров | Строковый тип фиксированной длины |

|  |  |  |
| --- | --- | --- |
| Имя идентификатора структуры | Назначение структуры | Тип структуры |
| Messagelength | Строка для хранения сообщений | Строковый тип фиксированной длины |
| Head\_Marshrutka | Заголовок списка маршруток | Глобальная переменная-указатель |
| Head\_Departure | Заголовок списка выездов | Глобальная переменная-указатель |
| Head\_Trip | Заголовок списка бронирований | Глобальная переменная-указатель |

Таблица 3.1 – продолжение

### 3.3. Описание структур в виде таблиц

Таблица 3.2 Используемые идентификаторы

|  |  |  |  |
| --- | --- | --- | --- |
| Имя идентификатора | Тип данных | Назначение | Структура |
| MarshrutkaNum | string[10] | Номер маршрутки | T\_Marshrutka |
| StartPlace | string[30] | Место отправления | T\_Marshrutka |
| FinishPlace | string[30] | Место прибытия | T\_Marshrutka |
| StartTime | string[5] | Время отправления | T\_Marshrutka |
| FinishTime | string[5] | Время прибытия | T\_Marshrutka |
| Price | real | Стоимость проезда | T\_Marshrutka |
| Seats | integer | Количество мест | T\_Marshrutka |
| info | T\_Marshrutka | Информация о маршрутке | T\_List\_Marshrutka |

Таблица 3.2 – Продолжение

|  |  |  |  |
| --- | --- | --- | --- |
| adr | T\_Pointer\_Marshrutka | Указатель на следующую маршрутку | T\_List\_Marshrutka |
| DepartureKey | integer | Код выезда | T\_Departure |
| MarshrutkaNum | string[10] | Номер маршрутки | T\_Departure |
| Date | string[5] | Дата выезда | T\_Departure |
| FreeSeats | integer | Количество свободных мест | T\_Departure |
| info | T\_Departure | Информация о выезде | T\_List\_Departure |
| BookKey | integer | Код бронирования | T\_Trip |
| DepartureKey | integer | Код выезда | T\_Trip |
| Name | string[30] | ФИО пассажира | T\_Trip |
| PhoneNumber | string[15] | Телефон пассажира | T\_Trip |
| Status | T\_Status | Статус бронирования | T\_Trip |
| info | T\_Trip | Информация о бронировании | T\_List\_Trip |
| LastDepartureKey | integer | Последний использованный код выезда | Глобальная переменная |
| LastBookKey | integer | Последний использованный код бронирования | Глобальная переменная |

### 

### 3.4. Определение подпрограмм и их описание

Таблица 3.3 Используемые идентификаторы

|  |  |  |  |
| --- | --- | --- | --- |
| Имя под- программы | Назначение подпрограммы | Заголовок подпрограммы | Имя параметра, его тип, назначение |
| SaveDataToFiles | Сохраняет все данные о маршрутках, выездах, бронированиях в файлы, вызывая процедуру на сохранение данных каждого типа | procedure SaveDataToFiles(Head\_Marshrutka, Head\_Departure, Head\_Trip) | Head\_Marshrutka: T\_Pointer\_Marshrutka — указатель на голову списка маршруток Head\_Departure: T\_Pointer\_Departure — указатель на голову списка выездов Head\_Trip: T\_Pointer\_Trip — указатель на голову списка бронирований |
| CreateTripInfoFile | Формирует текстовый файл с полной информацией о бронировании, маршрутке и выезде | procedure CreateTripInfoFile(TripInfo: T\_Trip; Head\_Departure, Head\_Marshrutka) | TripInfo: T\_Trip — данные о бронировании Head\_Departure: T\_Pointer\_Departure — указатель на голову списка выездов Head\_Marshrutka: T\_Pointer\_Marshrutka — указатель на голову списка маршруток |
| GetSeats | Ввод и проверка количества мест | procedure GetSeats(user\_message: Messagelength; var Seats: integer) | user\_message: Messagelength — подсказка var Seats: integer — возвращаемое число мест |
| ShowMarshrutkaRedactMenu | Меню выбора маршрутки для редактирования | procedure ShowMarshrutkaRedactMenu(Head\_Marshrutka: T\_Pointer\_Marshrutka) | Head\_Marshrutka: T\_Pointer\_Marshrutka — указатель на голову списка маршруток |
| ShowDepartureRedactMenu | Меню выбора выезда для редактирования | procedure ShowDepartureRedactMenu(Head\_Departure: T\_Pointer\_Departure) | Head\_Departure: T\_Pointer\_Departure — указатель на голову списка выездов |
| ShowTripRedactMenu | Меню выбора бронирования для редактирования | procedure ShowTripRedactMenu(Head\_Trip: T\_Pointer\_Trip) | Head\_Trip: T\_Pointer\_Trip — указатель на голову списка бронирований |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Имя под- программы | Назначение подпрограммы | Заголовок подпрограммы | Имя параметра, его тип, назначение |
| GetMarshrutkaNumber | Ввод и проверка номера маршрутки по формату, проверка на дублирование | procedure GetMarshrutkaNumber(user\_message: Messagelength; var MarshrutkaNum: Numberlength; check\_doubles: boolean) | user\_message: Messagelength — текст подсказки var MarshrutkaNum: Numberlength — возвращаемый номер check\_doubles: boolean — проверять ли на дубли |
| GetDate | Ввод и проверка корректности даты | procedure GetDate(user\_message: Messagelength; var Date: string) | user\_message: Messagelength — подсказка var Date: string — возвращаемая дата |
| GetTime | Ввод и проверка времени | procedure GetTime(user\_message: Messagelength; var Time: Timelength) | user\_message: Messagelength — подсказка var Time: Timelength — возвращаемое время |
| GetPrice | Ввод и проверка стоимости проезда | procedure GetPrice(user\_message: Messagelength; var Price: real) | user\_message: Messagelength — подсказка var Price: real — возвращаемая цена |

Таблица 3.3 - Продолжение

Таблица 3.3 - Продолжение

|  |  |  |  |
| --- | --- | --- | --- |
| Имя под- программы | Назначение подпрограммы | Заголовок подпрограммы | Имя параметра, его тип, назначение |
| AddMarshrutka | Добавляет новую маршрутку в список | procedure AddMarshrutka(Head\_Marshrutka: T\_Pointer\_Marshrutka) | Head\_Marshrutka: T\_Pointer\_Marshrutka — указатель на голову списка маршруток |
| ViewMarshrutkaList | Просмотр списка маршруток | procedure ViewMarshrutkaList(Head\_Marshrutka: T\_Pointer\_Marshrutka) | Head\_Marshrutka: T\_Pointer\_Marshrutka — указатель на голову списка маршруток |
| DeleteMarshrutka | Удаляет маршрутку из списка, если на неё нет выездов | function DeleteMarshrutka(Head\_Marshrutka: T\_Pointer\_Marshrutka; Head\_Departure: T\_Pointer\_Departure): boolean | Head\_Marshrutka: T\_Pointer\_Marshrutka — указатель на голову списка маршруток Head\_Departure: T\_Pointer\_Departure — указатель на голову списка выездов |
| AddDeparture | Добавляет новый выезд для выбранной маршрутки | procedure AddDeparture(Head\_Marshrutka: T\_Pointer\_Marshrutka; Head\_Departure: T\_Pointer\_Departure) | Head\_Marshrutka: T\_Pointer\_Marshrutka — указатель на голову списка маршруток |

Таблица 3.3 - Продолжение

|  |  |  |  |
| --- | --- | --- | --- |
| Имя под- программы | Назначение подпрограммы | Заголовок подпрограммы | Имя параметра, его тип, назначение |
| ViewDepartureList | Просмотр списка выездов | procedure ViewDepartureList(Head\_Departure: T\_Pointer\_Departure) | Head\_Departure: T\_Pointer\_Departure — указатель на голову списка выездов |
| DeleteDeparture | Удаляет выезд, если на него нет активных бронирований | function DeleteDeparture(Head\_Departure: T\_Pointer\_Departure; Head\_Trip: T\_Pointer\_Trip): boolean | Head\_Departure: T\_Pointer\_Departure — указатель на голову списка выездов Head\_Trip: T\_Pointer\_Trip — указатель на голову списка бронирований |
| AddTrip | Создаёт новое бронирование на выбранный выезд | procedure AddTrip(Head\_Marshrutka: T\_Pointer\_Marshrutka; Head\_Departure: T\_Pointer\_Departure; Head\_Trip: T\_Pointer\_Trip) | Head\_Marshrutka: T\_Pointer\_Marshrutka — указатель на голову списка маршруток Head\_Departure: T\_Pointer\_Departure — указатель на голову списка выездов Head\_Trip: T\_Pointer\_Trip — указатель на голову списка бронирований |

Таблица 3.3 - Продолжение

|  |  |  |  |
| --- | --- | --- | --- |
| Имя под- программы | Назначение подпрограммы | Заголовок подпрограммы | Имя параметра, его тип, назначение |
| ViewTripList | Просмотр списка бронирований | procedure ViewTripList(Head\_Trip: T\_Pointer\_Trip) | Head\_Trip: T\_Pointer\_Trip — указатель на голову списка бронирований |
| DeleteTrip | Удаляет бронирование, возвращает места в выезде | function DeleteTrip(Head\_Trip: T\_Pointer\_Trip; Head\_Departure: T\_Pointer\_Departure): boolean | Head\_Trip: T\_Pointer\_Trip — указатель на голову списка бронирований Head\_Departure: T\_Pointer\_Departure — указатель на голову списка выездов |
| SearchMarshrutka | Поиск маршрутки по пунктам и диапазону дат | procedure SearchMarshrutka(Head\_Marshrutka: T\_Pointer\_Marshrutka; Head\_Departure: T\_Pointer\_Departure; Head\_Trip: T\_Pointer\_Trip) | Head\_Marshrutka: T\_Pointer\_Marshrutka — указатель на голову списка маршруток Head\_Departure: T\_Pointer\_Departure — указатель на голову списка выездов Head\_Trip: T\_Pointer\_Trip — указатель на голову списка бронирований |

Таблица 3.3 - Продолжение

|  |  |  |  |
| --- | --- | --- | --- |
| Имя под- программы | Назначение подпрограммы | Заголовок подпрограммы | Имя параметра, его тип, назначение |
| EditTripName | Изменяет ФИО пассажира в бронировании | procedure EditTripName(Trip: T\_Pointer\_Trip) | Trip: T\_Pointer\_Trip — указатель на элемент бронирования |
| EditTripPhone | Изменяет номер телефона пассажира в бронировании | procedure EditTripPhone(Trip: T\_Pointer\_Trip) | Trip: T\_Pointer\_Trip — указатель на элемент бронирования |
| EditDepartureDate | Изменяет дату выезда (если нет бронирований на этот выезд) | procedure EditDepartureDate(Departure: T\_Pointer\_Departure) | Departure: T\_Pointer\_Departure — указатель на элемент выезда |
| EditDepartureFreeSeats | Изменяет количество свободных мест на выезде | procedure EditDepartureFreeSeats(Departure: T\_Pointer\_Departure) | Departure: T\_Pointer\_Departure — указатель на элемент выезда |
| EditMarshrutkaPlace | Изменяет место отправления или прибытия маршрутки | procedure EditMarshrutkaPlace(Marshrutka: T\_Pointer\_Marshrutka; IsStartPlace: boolean) | Marshrutka: T\_Pointer\_Marshrutka — указатель на элемент маршрутки IsStartPlace: boolean — если True, редактируется место отправления; если False — место прибытия |

Таблица 3.3 - Продолжение

|  |  |  |  |
| --- | --- | --- | --- |
| Имя под- программы | Назначение подпрограммы | Заголовок подпрограммы | Имя параметра, его тип, назначение |
| EditMarshrutkaTime | Изменяет время отправления или прибытия маршрутки | procedure EditMarshrutkaTime(Marshrutka: T\_Pointer\_Marshrutka; IsStartTime: boolean) | Marshrutka: T\_Pointer\_Marshrutka — указатель на элемент маршрутки IsStartTime: boolean — если True, редактируется время отправления; если False — время прибытия |
| EditMarshrutkaPrice | Изменяет цену проезда маршрутки | procedure EditMarshrutkaPrice(Marshrutka: T\_Pointer\_Marshrutka) | Marshrutka: T\_Pointer\_Marshrutka — указатель на элемент маршрутки |
| EditMarshrutkaSeats | Изменяет вместимость маршрутки | procedure EditMarshrutkaSeats(Marshrutka: T\_Pointer\_Marshrutka) | Marshrutka: T\_Pointer\_Marshrutka — указатель на элемент маршрутки |
| DisplayDepartureInfo | Показывает всю информацию о выезде | procedure DisplayDepartureInfo(Departure: T\_Pointer\_Departure) | Departure: T\_Pointer\_Departure — указатель на элемент выезда |
| EditDepartureDetails | Меню изменения параметров выезда | procedure EditDepartureDetails(Departure: T\_Pointer\_Departure) | Departure: T\_Pointer\_Departure — указатель на элемент выезда |

Таблица 3.3 – Продолжение

|  |  |  |  |
| --- | --- | --- | --- |
| EditTripStatus | Изменяет статус бронирования (подтверждён/не подтверждён) | procedure EditTripStatus(Trip: T\_Pointer\_Trip) | Trip: T\_Pointer\_Trip — указатель на элемент бронирования |
| DisplayMarshrutkaInfo | Показывает подробную информацию о маршрутке | procedure DisplayMarshrutkaInfo(Marshrutka: T\_Pointer\_Marshrutka) | Marshrutka: T\_Pointer\_Marshrutka — указатель на элемент маршрутки |
| EditMarshrutkaDetails | Меню изменения параметров маршрутки | procedure EditMarshrutkaDetails(Marshrutka: T\_Pointer\_Marshrutka) | Marshrutka: T\_Pointer\_Marshrutka — указатель на элемент маршрутки |
| EditTripDetails | Меню изменения параметров бронирования | procedure EditTripDetails(Trip: T\_Pointer\_Trip; Head\_Departure: T\_Pointer\_Departure; Head\_Marshrutka: T\_Pointer\_Marshrutka) | Trip: T\_Pointer\_Trip — указатель на элемент бронирования Head\_Departure: T\_Pointer\_Departure — указатель на голову списка выездов Head\_Marshrutka: T\_Pointer\_Marshrutka — указатель на голову списка маршруток |

Таблица 3.3 - Продолжение

|  |  |  |  |
| --- | --- | --- | --- |
| ShowMarshrutkaRedactMenu | Меню выбора маршрутки для редактирования | procedure ShowMarshrutkaRedactMenu(Head\_Marshrutka: T\_Pointer\_Marshrutka) | Head\_Marshrutka: T\_Pointer\_Marshrutka — указатель на голову списка маршруток |
| ShowDepartureRedactMenu | Меню выбора выезда для редактирования | procedure ShowDepartureRedactMenu(Head\_Departure: T\_Pointer\_Departure) | Head\_Departure: T\_Pointer\_Departure — указатель на голову списка выездов |
| ShowTripRedactMenu | Меню выбора бронирования для редактирования | procedure ShowTripRedactMenu(Head\_Trip: T\_Pointer\_Trip) | Head\_Trip: T\_Pointer\_Trip — указатель на голову списка бронирований |

### 3.5 Тестирование и отладка программы

Таблица 3.4 Прохождение тестов программы

| Номер теста | Вводимые данные | Ожидаемый результат | Полученный результат |
| --- | --- | --- | --- |
| 1 | Выбор основных действий главного меню программы (добавление, изменение, удаление) | Корректное открытие подменю каждого из действий. | Тест пройден. |
| 2 | Ввод в подменю добавления маршрутки корректных данных | Переход ко вводу информации о следующей характеристике маршрутки, пока не будут введены все данные. После ввода всех характеристик, сообщение о том, что маршрутка добавлена успешно. | Тест пройден. |
| 3 | Ввод в подменю добавления маршрутки некорректных данных (номера маршрутки неподходящей длины, стоимость записана не числом и т.д) | Вывод отладочного сообщения о том, что данная характеристика некорректна до тех пор, пока не будет введено её верное значение. Отладочное сообщение может сопровождаться указанием на конкретную ошибку ввода (неправильная длина номера). | Тест пройден. |
| 4 | Ввод некорректного номера, проходящего базовую проверку на длину (использование не предусмотренных символов, нарушение порядка расположения разрешённых символов) | Вывод отладочного сообщения о том, что номер некорректен до тех пор, пока он не будет введён корректно. | Тест пройден. |
| 5 | Попытка добавить маршрутку с номером, который уже существует в системе | Сообщение о существовании маршрутки с таким номером, требование повторить ввод. | Тест пройден. |
| 6 | Просмотр списка маршруток после добавления нескольких | Корректное отображение всех характеристик маршруток в виде таблицы | Тест пройден. |

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Вводимые данные | Ожидаемый результат | Полученный результат |
| 7 | Удаление маршрутки из списка, просмотр списка маршруток. | В списке маршруток теперь не отображается удалённая маршрутка | Тест пройден. |
| 8 | Ввод в подменю удаления номера маршрутки, которой не существует | Вывод отладочного сообщения о ненайденной маршрутке | Тест пройден. |
| 9 | Попытка удаления маршрутки при пустом списке | Вывод отладочного сообщения о пустом списке, выход в подменю удаления | Тест пройден. |
| 10 | Ввод всех корректных данных для выезда (номер маршрутки, дата, время, водитель и т.д.) | Пошаговый переход к следующему полю, после ввода всех данных — сообщение об успешном добавлении выезда | Тест пройден. |
| 11 | Ввод несуществующего номера маршрутки при добавлении выезда | Вывод сообщения о ненайденной маршрутке с описанным номером, возвращение в подменю добавления элемента в список. | Тест пройден. |
| 12 | Попытка удаления маршрутки, на которую назначен выезд | Вывод сообщения о невозможности удаления маршрутки. Пользователь возвращается в подменю удаления элементов списка. | Тест пройден. |
| 13 | Изменение информации маршрутки, на которую не назначено вызовов | Все характеристики маршрутки, вводимые пользователем доступны для изменения. После ввода новых данных выводится сообщение об успешном изменении маршрутки. | Тест пройден. |

Таблица 3.4 - Продолжение

# 

Таблица 3.4 - Продолжение

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Вводимые данные | Ожидаемый результат | Полученный результат |
| 14 | Попытка изменения информации маршрутки, на которую назначен выезд | Вывод сообщения о невозможности изменения. Пользователь возвращается в подменю изменения элементов списков. | Тест пройден. |
| 15 | Добавление нового бронирования на выбранный выезд | Переход ко вводу информации о следующей характеристике бронирования, пока не будут введены все данные. После ввода всех характеристик, сообщение о том, что бронирование добавлено успешно. В папке проекта появляется текстовый файл с информацией о бронировании. | Тест пройден. |
| 16 | Изменение данных существующего бронирования: Пользователь выбирает бронирование из списка и изменяет доступные параметры, в том числе невводимый при заказе бронирования параметр “статус бронирования”. | После завершения текстовый файл соответствующего бронирования перезаписывается. | Тест пройден. |
| 17 | Удаление выбранного бронирования: Пользователь выбирает нужное бронирование и подтверждает удаление. | После выполнения действия бронирование удаляется из списка, выводится сообщение об успешном удалении. | Тест пройден. |
| 18 | Изменение характеристик выезда, на который нет бронирований: Пользователь выбирает выезд, к которому не привязано ни одного бронирования, и изменяет любые параметры (дату, время, маршрутку и т.д.). | После ввода всех характеристик, сообщение о том, что выезд добавлен успешно. | Тест пройден. |

Таблица 3.4 - Продолжение

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Вводимые данные | Ожидаемый результат | Полученный результат |
| 19 | Попытка изменения характеристик выезда, на который есть бронирования | Вывод сообщения о невозможности изменения. Пользователь возвращается в подменю изменения элементов списков. | Тест пройден. |
| 20 | Выход с сохранением | При выборе опции выхода с сохранением система перезаписывает файлы departure.dat, marshrutka.dat, trip.dat, выводит уведомление об успешном сохранении, после чего происходит завершение работы программы. | Тест пройден. |
| 21 | Загрузка данных первым действием в программе | При запуске системы пользователь в меню выбирает загрузку данных из файла. После завершения загрузки все элементы (маршрутки, выезды, бронирования) корректно отображаются в соответствующих списках. | Тест пройден. |
| 22 | Попытка загрузки данных после работы с программой. Ввод нескольких маршруток, после чего выбор пункта загрузки данных. | Пользователь получает отладочное сообщение, о том, что данные можно загружать только в начале работы программы. | Тест пройден. |
| 23 | Открытие подменю сортировок и выбор сортировки по месту прибытия | Пользователь получает сообщение о том, что список отсортирован, после чего выводится список всех маршруток. | Тест пройден. |

Таблица 3.4 - Продолжение

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Вводимые данные | Ожидаемый результат | Полученный результат |
| 24 | Поиск маршрутки. Пользователь вводит пункт отправления, пункт прибытия, диапазон дат. | Пользователь получает информацию о всех маршрутках, подходящих под описанные условия. | Тест пройден. |
| 25 | Поиск маршрутки, результат которого пустой. | Пользователь получает сообщение о том, что маршруток не найдено. | Тест пройден. |

# 4. РУКОВОДСТВО ПО УСТАНОВКЕ И ЭКСПЛУАТАЦИИ

### 4.1. Установка программы

1. Системные требования:
   * Операционная система: Windows
   * Консольное приложение с минимальными требованиями к ресурсам системы
2. Процесс установки:
   * Программа не требует установки. Исполняемый файл (.exe) уже находится в папке программы
   * Для запуска программы достаточно открыть файл Upozn1.exe двойным щелчком мыши
   * Дополнительные файлы перед первым запуском не требуются
   * При первом запуске и последующем сохранении данных программа автоматически создаст необходимые типизированные файлы (departure.dat, marshrutka.dat, trip.dat) в директории программы
   * При создании бронирования будут генерироваться текстовые файлы с информацией о каждом бронировании

### 4.2. Эксплуатация программы

### 4.2.1. Запуск программы

После запуска программы пользователю будет представлено главное меню с следующими опциями:

1. Чтение данных из файла
2. Просмотр списка
3. Сортировка элементов списка
4. Поиск маршрутки
5. Добавление элементов списка
6. Удаление элементов списка
7. Изменение элементов списка
8. Забронировать место в маршрутке
9. Выход без сохранения
10. Выход с сохранением

### 4.2.2. Основные функции программы

Загрузка данных:

* Обратите внимание, что загрузка данных возможна только в начале работы программы
* Если файлы данных отсутствуют, программа начнет работу с пустыми списками

Просмотр данных:

* Пункт 2 позволяет просматривать списки маршруток, выездов и бронирований

Управление маршрутками:

* Добавление маршруток (пункт 5): ввод номера маршрутки, пунктов отправления и прибытия, времени и стоимости
* Редактирование маршруток (пункт 7): изменение любых параметров существующей маршрутки
* Удаление маршруток (пункт 6): удаление возможно только если маршрутка не используется в выездах

Управление выездами:

* Добавление выезда (пункт 5): указание маршрутки, даты выезда
* Редактирование выездов (пункт 7): изменение даты или количества свободных мест
* Удаление выездов (пункт 6): удаление возможно только если на выезд нет бронирований

Бронирование мест:

* Создание бронирования (пункт 8): выбор выезда, ввод данных пассажира и количества мест
* При создании бронирования автоматически генерируется текстовый файл с информацией о бронировании (Trip[номер].txt)
* Редактирование бронирований (пункт 7): изменение данных пассажира, количества мест или статуса бронирования
* Удаление бронирований (пункт 6): при удалении бронирования освобождаются места на соответствующем выезде

Поиск и сортировка:

* Поиск маршруток (пункт 4): поиск по пунктам отправления, прибытия и диапазону дат
* Сортировка данных (пункт 3): сортировка маршруток по месту прибытия, выездов по дате, бронирований по коду выезда

Сохранение данных:

* Для сохранения всех изменений перед выходом используйте пункт 10
* При выборе пункта 9 программа завершит работу без сохранения изменений
* Данные сохраняются в типизированные файлы departure.dat, marshrutka.dat и trip.dat в директории программы

### 4.2.3. Форматы ввода данных

Дата: формат ДД.ММ

* Дата должна быть действительной

Время: формат ЧЧ:ММ

* Время должно быть действительным

Номер маршрутки: два допустимых формата

* NNNNLL-N: четыре цифры, две буквы, дефис, одна цифра (например, 1234АВ-5)
* NLLLNNNN: цифра, три буквы, четыре цифры (например, 1АВС1234)
* Буквы должны быть из набора общих для белорусского и латинского алфавитов

Цена: положительное число с не более чем двумя знаками после десятичной точки

### 4.2.4. Работа с файлами

* Программа автоматически создает и обновляет типизированные файлы данных при сохранении (пункт 10 главного меню)
* При создании бронирования генерируется текстовый файл с информацией о бронировании (Trip[номер].txt)
* Файлы данных (departure.dat, marshrutka.dat, trip.dat) имеют бинарный формат и не предназначены для ручного редактирования

### 4.3. Возможные проблемы и их решения

* Проблема: Программа сообщает, что невозможно загрузить данные. Решение: Убедитесь, что файлы данных находятся в той же директории, что и исполняемый файл программы.
* Проблема: Невозможно удалить маршрутку. Решение: Маршрутка не может быть удалена, если для неё существуют выезды. Сначала удалите все связанные выезды.
* Проблема: Невозможно удалить выезд. Решение: Выезд не может быть удален, если на него есть бронирования. Сначала удалите все связанные бронирования.
* Проблема: Программа не находит маршрутки при поиске. Решение: Убедитесь, что вы вводите пункты отправления и прибытия точно так же, как они были введены при создании маршрутки (включая регистр букв).
* Проблема: Невозможно изменить характеристики выезда. Решение: Выезд, на который есть бронирования, не может быть изменен. Сначала удалите все связанные бронирования.

# Приложение 1. Код программы

program Upozn1;

uses System.SysUtils;

Type

Timelength = string[5];

Numberlength = string[10];

Messagelength = string[80];

T\_Status = (bsApprove, bsNotApprove);

T\_Marshrutka = record

MarshrutkaNum: string[10];

StartPlace: string[30];

FinishPlace: string[30];

StartTime: string[5];

FinishTime: string[5];

Price: real;

Seats: integer;

end;

T\_Pointer\_Marshrutka = ^T\_List\_Marshrutka;

T\_List\_Marshrutka = record

info: T\_Marshrutka;

adr: T\_Pointer\_Marshrutka;

end;

T\_Departure = record

DepartureKey: integer;

MarshrutkaNum: string[10];

Date: string[5];

FreeSeats: integer;

end;

T\_Pointer\_Departure = ^T\_List\_Departure;

T\_List\_Departure = record

info: T\_Departure;

adr: T\_Pointer\_Departure;

end;

T\_Trip = record

BookKey: integer;

DepartureKey: integer;

Name: string[30];

PhoneNumber: string[15];

Seats: integer;

Status: T\_status;

end;

T\_Pointer\_Trip = ^T\_List\_Trip;

T\_List\_Trip = record

info: T\_Trip;

adr: T\_Pointer\_Trip;

end;

var

Head\_Marshrutka: T\_Pointer\_Marshrutka;

Head\_Departure: T\_Pointer\_Departure;

Head\_Trip: T\_Pointer\_Trip;

LastDepartureKey, LastBookKey, choice, code: integer;

first\_open, isFirstAction: boolean;

choicestr: string;

procedure SaveDataToFiles (Head\_Marshrutka: T\_Pointer\_Marshrutka; Head\_Departure: T\_Pointer\_Departure; Head\_Trip: T\_Pointer\_Trip);

var

MarshrutkaFile: file of T\_Marshrutka;

DepartureFile: file of T\_Departure;

TripFile: file of T\_Trip;

CurrentMarshrutka: T\_Pointer\_Marshrutka;

CurrentDeparture: T\_Pointer\_Departure;

CurrentTrip: T\_Pointer\_Trip;

begin

AssignFile(MarshrutkaFile, 'marshrutka.dat');

Rewrite(MarshrutkaFile);

CurrentMarshrutka := Head\_Marshrutka^.adr;

while CurrentMarshrutka <> nil do

begin

Write(MarshrutkaFile, CurrentMarshrutka^.info);

CurrentMarshrutka := CurrentMarshrutka^.adr;

end;

CloseFile(MarshrutkaFile);

AssignFile(DepartureFile, 'departure.dat');

Rewrite(DepartureFile);

CurrentDeparture := Head\_Departure^.adr;

while CurrentDeparture <> nil do

begin

Write(DepartureFile, CurrentDeparture^.info);

CurrentDeparture := CurrentDeparture^.adr;

end;

CloseFile(DepartureFile);

AssignFile(TripFile, 'trip.dat');

Rewrite(TripFile);

CurrentTrip := Head\_Trip^.adr;

while CurrentTrip <> nil do

begin

Write(TripFile, CurrentTrip^.info);

CurrentTrip := CurrentTrip^.adr;

end;

CloseFile(TripFile);

WriteLn('Данные успешно сохранены в файлы.');

end;

procedure CreateTripInfoFile (TripInfo: T\_Trip; Head\_Departure: T\_Pointer\_Departure; Head\_Marshrutka: T\_Pointer\_Marshrutka);

var

InfoFile: TextFile;

FileName: string;

DepartureInfo: T\_Departure;

MarshrutkaInfo: T\_Marshrutka;

CurrentDeparture: T\_Pointer\_Departure;

CurrentMarshrutka: T\_Pointer\_Marshrutka;

StatusStr: string;

Found: boolean;

begin

FileName := 'Trip' + IntToStr(TripInfo.BookKey) + '.txt';

AssignFile(InfoFile, FileName);

Rewrite(InfoFile);

WriteLn(InfoFile, '=============== ИНФОРМАЦИЯ О БРОНИРОВАНИИ ===============');

WriteLn(InfoFile, 'Код бронирования: ', TripInfo.BookKey);

WriteLn(InfoFile, 'Код выезда: ', TripInfo.DepartureKey);

WriteLn(InfoFile, 'Имя: ', TripInfo.Name);

WriteLn(InfoFile, 'Количество мест: ', TripInfo.Seats);

WriteLn(InfoFile, 'Телефон: ', TripInfo.PhoneNumber);

case TripInfo.Status of

bsNotApprove: StatusStr := 'Не подтверждён';

bsApprove: StatusStr := 'Подтверждён';

end;

WriteLn(InfoFile, 'Статус: ', StatusStr);

WriteLn(InfoFile);

// Ищем информацию о выезде

Found := false;

CurrentDeparture := Head\_Departure^.adr;

while (CurrentDeparture <> nil) and (not Found) do

begin

if CurrentDeparture^.Info.DepartureKey = TripInfo.DepartureKey then

begin

DepartureInfo := CurrentDeparture^.Info;

Found := true;

end;

CurrentDeparture := CurrentDeparture^.adr;

end;

WriteLn(InfoFile, 'Дата выезда: ', DepartureInfo.Date);

WriteLn(InfoFile);

// Ищем информацию о маршрутке

Found := false;

CurrentMarshrutka := Head\_Marshrutka^.adr;

while (CurrentMarshrutka <> nil) and (not Found) do

begin

if CurrentMarshrutka^.Info.MarshrutkaNum = DepartureInfo.MarshrutkaNum then

begin

MarshrutkaInfo := CurrentMarshrutka^.Info;

Found := true;

end;

CurrentMarshrutka := CurrentMarshrutka^.adr;

end;

WriteLn(InfoFile, 'Место отправления: ', MarshrutkaInfo.StartPlace);

WriteLn(InfoFile, 'Место прибытия: ', MarshrutkaInfo.FinishPlace);

WriteLn(InfoFile, 'Время отправления: ', MarshrutkaInfo.StartTime);

WriteLn(InfoFile, 'Время прибытия: ', MarshrutkaInfo.FinishTime);

WriteLn(InfoFile, 'Стоимость одного места: ', MarshrutkaInfo.Price:0:2, ' руб.');

CloseFile(InfoFile);

end;

procedure GetMarshrutkaNumber(user\_message: Messagelength; var MarshrutkaNum: Numberlength; check\_doubles: boolean);

var

isValid: boolean;

Current: T\_Pointer\_Marshrutka;

i: integer;

ValidLetters: set of char;

isFirstFormatValid, isSecondFormatValid: boolean;

begin

//буквы общие для белорусского и латинского алфавитов

ValidLetters := ['A', 'B', 'E', 'I', 'K', 'M', 'H', 'O', 'P', 'C', 'T', 'X',

'А', 'В', 'Е', 'І', 'К', 'М', 'Н', 'О', 'Р', 'С', 'Т', 'Х'];

repeat

WriteLn(user\_message);

ReadLn(MarshrutkaNum);

isValid := True;

if Length(MarshrutkaNum) <> 8 then

begin

WriteLn('Ошибка: Номер маршрутки должен состоять ровно из 8 символов.');

isValid := False;

end

else

begin

// Проверка первого формата: NNNNLL-N

isFirstFormatValid := True;

// Проверка позиций 1-4

i := 1;

while (i <= 4) and isFirstFormatValid do

begin

if not (MarshrutkaNum[i] in ['0'..'9']) then

isFirstFormatValid := False;

Inc(i);

end;

// Проверка позиций 5-6

if isFirstFormatValid then

begin

if not ((MarshrutkaNum[5] in ValidLetters) and (MarshrutkaNum[6] in ValidLetters)) then

isFirstFormatValid := False;

end;

// Проверка позиции 7

if isFirstFormatValid then

begin

if MarshrutkaNum[7] <> '-' then

isFirstFormatValid := False;

end;

// Проверка позиции 8

if isFirstFormatValid then

begin

if not (MarshrutkaNum[8] in ['1'..'8']) then

isFirstFormatValid := False;

end;

// Проверка второго формата: NLLLNNNN

isSecondFormatValid := True;

if not (MarshrutkaNum[1] in ['1'..'8']) then

isSecondFormatValid := False;

if isSecondFormatValid then

begin

i := 2;

while (i <= 4) and isSecondFormatValid do

begin

if not (MarshrutkaNum[i] in ValidLetters) then

isSecondFormatValid := False;

Inc(i);

end;

end;

if isSecondFormatValid then

begin

i := 5;

while (i <= 8) and isSecondFormatValid do

begin

if not (MarshrutkaNum[i] in ['0'..'9']) then

isSecondFormatValid := False;

Inc(i);

end;

end;

if not (isFirstFormatValid or isSecondFormatValid) then

begin

WriteLn('Ошибка: Номер маршрутки не соответствует ни одному из допустимых форматов.');

isValid := False;

end

else

if check\_doubles = True then

begin

Current := Head\_Marshrutka;

while (Current <> nil) and isValid do

begin

if Current^.info.MarshrutkaNum = MarshrutkaNum then

begin

WriteLn('Ошибка: Маршрутка с таким номером уже существует.');

isValid := False;

end;

Current := Current^.adr;

end;

end;

end;

until isValid;

end;

procedure GetDate(user\_message: Messagelength; var Date: string);

var

isValid: boolean;

Day, Month, Code: integer;

DayStr, MonthStr: string;

begin

repeat

WriteLn(user\_message);

ReadLn(Date);

isValid := True;

if (Length(Date) <> 5) or (Date[3] <> '.') then

begin

WriteLn('Ошибка: Неверный формат даты. Используйте формат ДД.ММ');

isValid := False;

end

else

begin

DayStr := Copy(Date, 1, 2);

MonthStr := Copy(Date, 4, 2);

Val(DayStr, Day, Code);

if Code <> 0 then

begin

WriteLn('Ошибка: День должен быть числом.');

isValid := False;

end;

Val(MonthStr, Month, Code);

if Code <> 0 then

begin

WriteLn('Ошибка: Месяц должен быть числом.');

isValid := False;

end;

if isValid then

begin

if (Month < 1) or (Month > 12) then

begin

WriteLn('Ошибка: Месяц должен быть от 1 до 12.');

isValid := False;

end

else if (Day < 1) or (Day > 31) then

begin

WriteLn('Ошибка: День должен быть от 1 до 31.');

isValid := False;

end

else if ((Month = 4) or (Month = 6) or (Month = 9) or (Month = 11)) and (Day > 30) then

begin

WriteLn('Ошибка: В этом месяце не может быть больше 30 дней.');

isValid := False;

end

else if (Month = 2) and (Day > 29) then

begin

WriteLn('Ошибка: В феврале не может быть больше 29 дней.');

isValid := False;

end;

end;

end;

until isValid;

end;

procedure GetTime(user\_message: Messagelength; var Time: Timelength);

var

isValid: boolean;

Hour, Minute, Code: integer;

HourStr, MinuteStr: string;

TempTime: string;

begin

repeat

WriteLn(user\_message);

ReadLn(TempTime);

isValid := True;

if (Length(TempTime) <> 5) or (TempTime[3] <> ':') then

begin

WriteLn('Ошибка: Неверный формат времени. Используйте формат ЧЧ:ММ');

isValid := False;

end

else

begin

HourStr := Copy(TempTime, 1, 2);

MinuteStr := Copy(TempTime, 4, 2);

Val(HourStr, Hour, Code);

if Code <> 0 then

begin

WriteLn('Ошибка: Часы должны быть числом.');

isValid := False;

end;

Val(MinuteStr, Minute, Code);

if Code <> 0 then

begin

WriteLn('Ошибка: Минуты должны быть числом.');

isValid := False;

end;

if isValid then

begin

if (Hour < 0) or (Hour > 23) then

begin

WriteLn('Ошибка: Часы должны быть от 0 до 23.');

isValid := False;

end

else if (Minute < 0) or (Minute > 59) then

begin

WriteLn('Ошибка: Минуты должны быть от 0 до 59.');

isValid := False;

end;

end;

end;

if isValid then

Time := TempTime;

until isValid;

end;

procedure GetPrice(user\_message: Messagelength; var Price: real);

var

PriceStr: string;

Code: integer;

isValid: boolean;

DotPos: integer;

FractionalPart: string;

begin

repeat

WriteLn(user\_message);

ReadLn(PriceStr);

for DotPos := 1 to Length(PriceStr) do

if PriceStr[DotPos] = ',' then

PriceStr[DotPos] := '.';

isValid := True;

DotPos := Pos('.', PriceStr);

if DotPos > 0 then

begin

FractionalPart := Copy(PriceStr, DotPos + 1, Length(PriceStr) - DotPos);

if Length(FractionalPart) > 2 then

begin

WriteLn('Ошибка: После десятичной точки должно быть не более двух цифр.');

isValid := False;

end;

end;

if isValid then

begin

Val(PriceStr, Price, Code);

isValid := (Code = 0) and (Price > 0);

if not isValid then

WriteLn('Ошибка: Введите корректную положительную стоимость.');

end;

until isValid;

end;

procedure GetSeats(user\_message: Messagelength; var Seats: integer);

var

SeatsStr: string;

Code: integer;

isValid: boolean;

begin

repeat

WriteLn(user\_message);

ReadLn(SeatsStr);

Val(SeatsStr, Seats, Code);

isValid := (Code = 0) and (Seats > 0);

if not isValid then

WriteLn('Ошибка: Введите корректное положительное количество мест.');

until isValid;

end;

procedure GetFreeSeats(user\_message: Messagelength; var Seats: integer);

var

SeatsStr: string;

Code: integer;

isValid: boolean;

begin

repeat

WriteLn(user\_message);

ReadLn(SeatsStr);

Val(SeatsStr, Seats, Code);

isValid := (Code = 0) and (Seats > 0);

if not isValid then

WriteLn('Ошибка: Введите корректное положительное количество мест.');

until isValid;

end;

procedure AddMarshrutka(Head\_Marshrutka: T\_Pointer\_Marshrutka);

var

NewMarshrutka: T\_Marshrutka;

NewNode: T\_Pointer\_Marshrutka;

Current: T\_Pointer\_Marshrutka;

begin

GetMarshrutkaNumber('Введите номер маршрутки:', NewMarshrutka.MarshrutkaNum, True);

WriteLn('Введите пункт отправления:');

ReadLn(NewMarshrutka.StartPlace);

WriteLn('Введите пункт назначения:');

ReadLn(NewMarshrutka.FinishPlace);

GetTime('Введите время отправления (ЧЧ:ММ):', NewMarshrutka.StartTime);

GetTime('Введите время прибытия (ЧЧ:ММ):', NewMarshrutka.FinishTime);

GetPrice('Введите стоимость проезда (в белорусских рублях):', NewMarshrutka.Price);

GetSeats('Введите количество мест в маршрутке:', NewMarshrutka.Seats);

New(NewNode);

NewNode^.info := NewMarshrutka;

NewNode^.adr := nil;

if Head\_Marshrutka^.adr = nil then

Head\_Marshrutka^.adr := NewNode

else

begin

Current := Head\_Marshrutka^.adr;

while Current^.adr <> nil do

Current := Current^.adr;

Current^.adr := NewNode;

end;

WriteLn('Маршрутка успешно добавлена!');

end;

procedure ViewMarshrutkaList (Head\_Marshrutka: T\_Pointer\_Marshrutka);

var

Current: T\_Pointer\_Marshrutka;

Count: integer;

begin

if Head\_Marshrutka^.adr = nil then

begin

WriteLn('Список маршруток пуст.');

end

else

begin

WriteLn('=== СПИСОК МАРШРУТОК ===');

Current := Head\_Marshrutka^.adr;

Count := 0;

WriteLn('--------------------------------------------------------------------------------------------------');

WriteLn('| Номер | Откуда | Куда | Отправление | Прибытие | Цена | Мест |');

WriteLn('--------------------------------------------------------------------------------------------------');

while Current <> nil do

begin

WriteLn('| ', Current^.info.MarshrutkaNum:10, ' | ', Current^.info.StartPlace:18, ' | ', Current^.info.FinishPlace:18, ' | ',

Current^.info.StartTime:6, ' | ', Current^.info.FinishTime:6,' | ',

Current^.info.Price:4:2, ' | ', Current^.info.Seats:3, ' |');

Current := Current^.adr;

Inc(Count);

end;

WriteLn('--------------------------------------------------------------------------------------------------');

WriteLn('Всего маршруток: ', Count);

end;

end;

function DeleteMarshrutka(Head\_Marshrutka: T\_Pointer\_Marshrutka; Head\_Departure: T\_Pointer\_Departure): boolean;

var

Current, Previous: T\_Pointer\_Marshrutka;

DepartureCurrent: T\_Pointer\_Departure;

Found, HasDepartures: boolean;

MarshrutkaId: Numberlength;

begin

ViewMarshrutkaList(Head\_Marshrutka);

Result := False;

Found := False;

HasDepartures := False;

if Head\_Marshrutka^.adr = nil then

WriteLn('Список маршруток пуст. Нечего удалять.')

else

begin

GetMarshrutkaNumber('Введите номер маршрутки для удаления:', MarshrutkaId, False);

DepartureCurrent := Head\_Departure^.adr;

while (DepartureCurrent <> nil) and (not HasDepartures) do

begin

if DepartureCurrent^.info.MarshrutkaNum = MarshrutkaId then

HasDepartures := True

else

DepartureCurrent := DepartureCurrent^.adr;

end;

if HasDepartures then

begin

WriteLn('Невозможно удалить маршрутку, так как для неё существуют выезды.');

end

else

begin

Current := Head\_Marshrutka^.adr;

Previous := Head\_Marshrutka;

while (Current <> nil) and (not Found) do

begin

if Current^.info.MarshrutkaNum = MarshrutkaId then

Found := True

else

begin

Previous := Current;

Current := Current^.adr;

end;

end;

// Если элемент найден, удаляем его

if Found then

begin

Previous^.adr := Current^.adr;

Dispose(Current);

WriteLn('Маршрутка успешно удалена.');

Result := True;

end

else

begin

WriteLn('Маршрутка с номером "', MarshrutkaId, '" не найдена.');

end;

end;

end;

end;

procedure AddDeparture (Head\_Marshrutka: T\_Pointer\_Marshrutka; Head\_Departure: T\_Pointer\_Departure);

var

NewDeparture: T\_Pointer\_Departure;

MarshrutkaNum: string;

FoundMarshrutka: boolean;

DateStr: string;

MaxSeats: integer;

Current: T\_Pointer\_Marshrutka;

Last: T\_Pointer\_Departure;

begin

Write('Введите номер маршрутки: ');

ReadLn(MarshrutkaNum);

FoundMarshrutka := False;

Current := Head\_Marshrutka^.adr;

MaxSeats := 0;

while (Current <> nil) and (not FoundMarshrutka) do

begin

if Current^.info.MarshrutkaNum = MarshrutkaNum then

begin

FoundMarshrutka := True;

MaxSeats := Current^.info.Seats;

end

else

Current := Current^.adr;

end;

if FoundMarshrutka then

begin

GetDate('Введите дату выезда маршрутки:', DateStr);

New(NewDeparture);

Inc(LastDepartureKey);

NewDeparture^.info.DepartureKey := LastDepartureKey;

NewDeparture^.info.MarshrutkaNum := MarshrutkaNum;

NewDeparture^.info.Date := DateStr;

NewDeparture^.info.FreeSeats := MaxSeats;

NewDeparture^.adr := nil;

if Head\_Departure^.adr = nil then

Head\_Departure^.adr := NewDeparture

else

begin

Last := Head\_Departure^.adr;

while Last^.adr <> nil do

Last := Last^.adr;

Last^.adr := NewDeparture;

end;

WriteLn('Выезд успешно добавлен. Код выезда: ', LastDepartureKey);

WriteLn('Количество свободных мест: ', MaxSeats);

end

else

begin

WriteLn('Маршрутка с номером ', MarshrutkaNum, ' не найдена.');

end;

end;

procedure ViewDepartureList (Head\_Departure: T\_Pointer\_Departure);

var

Current: T\_Pointer\_Departure;

Count: integer;

begin

if Head\_Departure^.adr = nil then

begin

WriteLn('Список выездов пуст.');

end

else

begin

Current := Head\_Departure^.adr;

Count := 0;

WriteLn('------------------------------------------------');

WriteLn('| Код | № маршрутки | Дата | Свободно |');

WriteLn('------------------------------------------------');

while Current <> nil do

begin

WriteLn('| ', Current^.info.DepartureKey:10, ' | ', Current^.info.MarshrutkaNum:10, ' | ',

Current^.info.Date:6, ' | ', Current^.info.FreeSeats:5, ' |');

Current := Current^.adr;

Inc(Count);

end;

WriteLn('------------------------------------------------');

WriteLn('Всего выездов: ', Count);

end;

end;

function DeleteDeparture (Head\_Departure: T\_Pointer\_Departure; Head\_Trip: T\_Pointer\_Trip): boolean;

var

Current, Previous: T\_Pointer\_Departure;

TripPtr: T\_Pointer\_Trip;

Found: boolean;

HasActiveTrips: boolean;

DepartureIdStr: string;

DepartureId: integer;

Code: integer;

ValidInput: boolean;

begin

ViewDepartureList (Head\_Departure);

Result := False;

Found := False;

HasActiveTrips := False;

ValidInput := False;

if Head\_Departure^.adr = nil then

WriteLn('Список выездов пуст. Нечего удалять.')

else

begin

repeat

WriteLn('Введите код выезда для удаления:');

ReadLn(DepartureIdStr);

Val(DepartureIdStr, DepartureId, Code);

if Code = 0 then

ValidInput := True

else

WriteLn('Ошибка: Код выезда должен быть числом.');

until ValidInput;

TripPtr := Head\_Trip^.adr;

while TripPtr <> nil do

begin

if (TripPtr^.info.DepartureKey = DepartureId) then

begin

HasActiveTrips := True;

end;

TripPtr := TripPtr^.adr;

end;

if HasActiveTrips then

begin

WriteLn('Невозможно удалить выезд, так как на него есть активные бронирования.');

Result := False;

end

else

begin

Current := Head\_Departure^.adr;

Previous := Head\_Departure;

while (Current <> nil) and (not Found) do

begin

if Current^.info.DepartureKey = DepartureId then

Found := True

else

begin

Previous := Current;

Current := Current^.adr;

end;

end;

if Found then

begin

Previous^.adr := Current^.adr;

Dispose(Current);

WriteLn('Выезд с кодом ', DepartureId, ' успешно удален.');

Result := True;

end

else

begin

WriteLn('Выезд с кодом ', DepartureId, ' не найден.');

end;

end;

end;

end;

procedure AddTrip (Head\_Marshrutka: T\_Pointer\_Marshrutka; Head\_Departure: T\_Pointer\_Departure; Head\_Trip: T\_Pointer\_Trip);

var

NewTrip: T\_Pointer\_Trip;

DepartureKey: string;

DepartureKeyInt: integer;

FoundDeparture: boolean;

Current: T\_Pointer\_Departure;

CurrentTrip: T\_Pointer\_Trip;

SeatsToBook: string;

SeatsToBookInt: integer;

Code: integer;

ValidInput: boolean;

begin

WriteLn('Доступные выезды:');

ViewDepartureList (Head\_Departure);

WriteLn('Введите код выезда для бронирования:');

ReadLn(DepartureKey);

Val(DepartureKey, DepartureKeyInt, Code);

if Code = 0 then

begin

FoundDeparture := False;

Current := Head\_Departure^.adr;

while (Current <> nil) and (not FoundDeparture) do

begin

if Current^.info.DepartureKey = DepartureKeyInt then

FoundDeparture := True

else

Current := Current^.adr;

end;

if FoundDeparture then

begin

if Current^.info.FreeSeats > 0 then

begin

New(NewTrip);

Inc(LastBookKey);

NewTrip^.info.BookKey := LastBookKey;

NewTrip^.info.DepartureKey := DepartureKeyInt;

WriteLn('Введите имя пассажира:');

ReadLn(NewTrip^.info.Name);

WriteLn('Введите номер телефона:');

ReadLn(NewTrip^.info.PhoneNumber);

ValidInput := False;

repeat

WriteLn('Введите количество бронируемых мест (доступно: ', Current^.info.FreeSeats, '):');

ReadLn(SeatsToBook);

Val(SeatsToBook, SeatsToBookInt, Code);

if Code = 0 then

begin

// Проверяем, что число положительное и не превышает доступное количество мест

if (SeatsToBookInt > 0) and (SeatsToBookInt <= Current^.info.FreeSeats) then

ValidInput := True

else if (SeatsToBookInt <= 0) then

WriteLn('Ошибка: Количество мест должно быть положительным числом.')

else

WriteLn('Ошибка: Недостаточно свободных мест. Доступно: ', Current^.info.FreeSeats);

end

else

WriteLn('Ошибка: Количество мест должно быть числом.');

until ValidInput;

NewTrip^.info.Seats := SeatsToBookInt;

NewTrip^.info.Status := bsNotApprove;

Current^.info.FreeSeats := Current^.info.FreeSeats - SeatsToBookInt;

NewTrip^.adr := nil;

if Head\_Trip^.adr = nil then

Head\_Trip^.adr := NewTrip

else

begin

CurrentTrip := Head\_Trip^.adr;

while CurrentTrip^.adr <> nil do

CurrentTrip := CurrentTrip^.adr;

CurrentTrip^.adr := NewTrip;

end;

WriteLn('Бронирование успешно создано!');

WriteLn('Код бронирования: ', LastBookKey);

WriteLn('Количество забронированных мест: ', SeatsToBookInt);

WriteLn('Осталось свободных мест на выезде: ', Current^.info.FreeSeats);

CreateTripInfoFile(NewTrip^.info, Head\_Departure, Head\_Marshrutka);

end

else

WriteLn('На этот выезд нет свободных мест.');

end

else

WriteLn('Выезд с кодом ', DepartureKeyInt, ' не найден.');

end

else

WriteLn('Ошибка: Код выезда должен быть числом.');

end;

procedure ViewTripList (Head\_Trip: T\_Pointer\_Trip);

var

Current: T\_Pointer\_Trip;

Count: integer;

StatusStr: string;

begin

if Head\_Trip^.adr = nil then

begin

WriteLn('Список бронирований пуст.');

end

else

begin

Current := Head\_Trip^.adr;

Count := 0;

WriteLn('--------------------------------------------------------------------------------------------');

WriteLn('| Код | Код выезда | Имя | Места | Телефон | Статус |');

WriteLn('--------------------------------------------------------------------------------------------');

while Current <> nil do

begin

case Current^.info.status of

bsNotApprove: StatusStr := 'Не подтверждён';

bsApprove: StatusStr := 'Подтверждён';

end;

WriteLn('| ', Current^.info.BookKey:7, ' | ', Current^.info.DepartureKey:10, ' | ', Current^.info.Name:15, ' | ',

Current^.info.Seats:5, ' | ', Current^.info.PhoneNumber:15, ' | ', StatusStr:17, ' |');

Current := Current^.adr;

Inc(Count);

end;

WriteLn('--------------------------------------------------------------------------------------------');

WriteLn('Всего бронирований: ', Count);

end;

end;

function DeleteTrip (Head\_Trip: T\_Pointer\_Trip; Head\_Departure: T\_Pointer\_Departure): boolean;

var

Current, Previous: T\_Pointer\_Trip;

DepartureCurrent: T\_Pointer\_Departure;

DepartureFound, ValidInput, Found: boolean;

TripIdStr: string;

TripId, Code: integer;

begin

Result := False;

Found := False;

ValidInput := False;

if Head\_Trip^.adr = nil then

WriteLn('Список бронирований пуст. Нечего удалять.')

else

begin

repeat

ViewTripList (Head\_Trip);

WriteLn('Введите код бронирования для удаления:');

ReadLn(TripIdStr);

Val(TripIdStr, TripId, Code);

if Code = 0 then

ValidInput := True

else

WriteLn('Ошибка: Код выезда должен быть числом.');

until ValidInput;

Current := Head\_Trip^.adr;

Previous := Head\_Trip;

while (Current <> nil) and (not Found) do

begin

if Current^.info.BookKey = TripId then

Found := True

else

begin

Previous := Current;

Current := Current^.adr;

end;

end;

if Found then

begin

DepartureCurrent := Head\_Departure^.adr;

DepartureFound := False;

while (DepartureCurrent <> nil) and (not DepartureFound) do

begin

if DepartureCurrent^.info.DepartureKey = Current^.info.DepartureKey then

begin

Inc(DepartureCurrent^.info.FreeSeats, Current^.info.Seats);

DepartureFound := True;

end

else

DepartureCurrent := DepartureCurrent^.adr;

end;

Previous^.adr := Current^.adr;

Dispose(Current);

WriteLn('Бронирование с кодом ', TripId, ' успешно удалено.');

Result := True;

end

else

begin

WriteLn('Бронирование с кодом ', TripId, ' не найдено.');

end;

end;

end;

procedure SearchMarshrutka (Head\_Marshrutka: T\_Pointer\_Marshrutka; Head\_Departure: T\_Pointer\_Departure; Head\_Trip: T\_Pointer\_Trip);

var

StartPlace, FinishPlace: string;

StartDate, EndDate: string;

CurrentMarshrutka: T\_Pointer\_Marshrutka;

CurrentDeparture: T\_Pointer\_Departure;

FoundResults: Boolean;

function IsDateInRange(DateToCheck, StartDate, EndDate: string): Boolean;

var

CheckDay, CheckMonth: Integer;

StartDay, StartMonth: Integer;

EndDay, EndMonth: Integer;

Code: Integer;

begin

Val(Copy(DateToCheck, 1, 2), CheckDay, Code);

Val(Copy(DateToCheck, 4, 2), CheckMonth, Code);

Val(Copy(StartDate, 1, 2), StartDay, Code);

Val(Copy(StartDate, 4, 2), StartMonth, Code);

Val(Copy(EndDate, 1, 2), EndDay, Code);

Val(Copy(EndDate, 4, 2), EndMonth, Code);

Result := ((CheckMonth > StartMonth) or ((CheckMonth = StartMonth) and (CheckDay >= StartDay))) and

((CheckMonth < EndMonth) or ((CheckMonth = EndMonth) and (CheckDay <= EndDay)));

end;

begin

WriteLn('Введите пункт отправления:');

ReadLn(StartPlace);

WriteLn('Введите пункт прибытия:');

ReadLn(FinishPlace);

GetDate('Введите начальную дату диапазона:', StartDate);

GetDate('Введите конечную дату диапазона:', EndDate);

FoundResults := False;

CurrentMarshrutka := Head\_Marshrutka^.adr;

while CurrentMarshrutka <> nil do

begin

if (CurrentMarshrutka^.info.StartPlace = StartPlace) and (CurrentMarshrutka^.info.FinishPlace = FinishPlace) then

begin

CurrentDeparture := Head\_Departure^.adr;

while CurrentDeparture <> nil do

begin

if (CurrentDeparture^.info.MarshrutkaNum = CurrentMarshrutka^.info.MarshrutkaNum) and IsDateInRange(CurrentDeparture^.info.Date, StartDate, EndDate) then

begin

if not FoundResults then

begin

WriteLn('Результаты поиска:');

WriteLn('--------------------------------------------------');

WriteLn('| Номер такси | Дата | Время отправления |');

WriteLn('--------------------------------------------------');

FoundResults := True;

end;

WriteLn('| ', CurrentDeparture^.info.MarshrutkaNum:11, ' | ', CurrentDeparture^.info.Date:8, ' | ', CurrentMarshrutka^.info.StartTime:20, ' |');

end;

CurrentDeparture := CurrentDeparture^.adr;

end;

end;

CurrentMarshrutka := CurrentMarshrutka^.adr;

end;

if FoundResults then

WriteLn('--------------------------------------------------')

else

WriteLn('Не найдено такси по указанному маршруту в выбранный диапазон дат.');

end;

procedure EditTripName(Trip: T\_Pointer\_Trip);

var

NewName: string;

ValidInput: boolean;

begin

WriteLn;

WriteLn('Текущее ФИО пассажира: ', Trip^.info.Name);

ValidInput := False;

while not ValidInput do

begin

Write('Введите новое ФИО пассажира: ');

ReadLn(NewName);

if NewName <> '' then

begin

Trip^.info.Name := NewName;

ValidInput := True;

end

else

WriteLn('ФИО пассажира не может быть пустым. Попробуйте снова.');

end;

end;

procedure EditTripPhone(Trip: T\_Pointer\_Trip);

var

NewPhone: string;

ValidInput: boolean;

begin

WriteLn;

WriteLn('Текущий номер телефона: ', Trip^.info.PhoneNumber);

ValidInput := False;

while not ValidInput do

begin

Write('Введите новый номер телефона: ');

ReadLn(NewPhone);

if NewPhone <> '' then

begin

Trip^.info.PhoneNumber := NewPhone;

ValidInput := True;

end

else

WriteLn('Номер телефона не может быть пустым. Попробуйте снова.');

end;

end;

procedure EditDepartureDate(Departure: T\_Pointer\_Departure);

var

NewDate: string;

CurrentTrip: T\_Pointer\_Trip;

HasBookings: boolean;

begin

CurrentTrip := Head\_Trip^.adr;

HasBookings := False;

while (CurrentTrip <> nil) and (not HasBookings) do

begin

if CurrentTrip^.info.DepartureKey = Departure^.info.DepartureKey then

HasBookings := True

else

CurrentTrip := CurrentTrip^.adr;

end;

if HasBookings then

begin

WriteLn('Невозможно изменить дату выезда, так как на него есть бронирования.');

end

else

begin

GetDate('Введите новую дату выезда:', NewDate);

Departure^.info.Date := NewDate;

WriteLn('Дата выезда успешно изменена на: ', NewDate);

end;

end;

procedure EditDepartureFreeSeats(Departure: T\_Pointer\_Departure);

var

NewSeats, TotalBookedSeats, TotalMarshrutkaSeats: integer;

CurrentTrip: T\_Pointer\_Trip;

CurrentMarshrutka: T\_Pointer\_Marshrutka;

Found: boolean;

begin

WriteLn;

WriteLn('Текущее количество свободных мест: ', Departure^.info.FreeSeats);

CurrentMarshrutka := Head\_Marshrutka^.adr;

TotalMarshrutkaSeats := 0;

Found := false;

while (CurrentMarshrutka <> nil) and (not Found) do

begin

if CurrentMarshrutka^.info.MarshrutkaNum = Departure^.info.MarshrutkaNum then

begin

TotalMarshrutkaSeats := CurrentMarshrutka^.info.Seats;

Found := true;

end;

if not Found then

CurrentMarshrutka := CurrentMarshrutka^.adr;

end;

TotalBookedSeats := 0;

CurrentTrip := Head\_Trip^.adr;

while CurrentTrip <> nil do

begin

if CurrentTrip^.info.DepartureKey = Departure^.info.DepartureKey then

TotalBookedSeats := TotalBookedSeats + CurrentTrip^.info.Seats;

CurrentTrip := CurrentTrip^.adr;

end;

GetFreeSeats('Введите новое количество свободных мест:', NewSeats);

if NewSeats + TotalBookedSeats <= TotalMarshrutkaSeats then

begin

Departure^.info.FreeSeats := NewSeats;

WriteLn('Количество свободных мест успешно обновлено.');

end

else

WriteLn('Ошибка: Невозможно установить такое количество свободных мест, так как сумма мест превышает вместимость маршрутки.');

end;

procedure EditMarshrutkaPlace(Marshrutka: T\_Pointer\_Marshrutka; IsStartPlace: boolean);

var

NewPlace: string;

CanEdit: boolean;

CurrentDeparture: T\_Pointer\_Departure;

begin

CanEdit := True;

CurrentDeparture := Head\_Departure;

while (CurrentDeparture <> nil) and CanEdit do

begin

if CurrentDeparture^.info.MarshrutkaNum = Marshrutka^.info.MarshrutkaNum then

begin

WriteLn('Невозможно изменить маршрут. Маршрутка уже используется в выезде с кодом: ',

CurrentDeparture^.info.DepartureKey);

CanEdit := False;

end;

CurrentDeparture := CurrentDeparture^.adr;

end;

if CanEdit then

begin

Write('Введите новое место ');

if IsStartPlace then

Write('отправления: ')

else

Write('прибытия: ');

ReadLn(NewPlace);

if NewPlace <> '' then

begin

if IsStartPlace then

Marshrutka^.info.StartPlace := NewPlace

else

Marshrutka^.info.FinishPlace := NewPlace;

WriteLn('Информация успешно обновлена.');

end

else

WriteLn('Поле не может быть пустым. Изменения не сохранены.');

end;

end;

procedure EditMarshrutkaTime(Marshrutka: T\_Pointer\_Marshrutka; IsStartTime: boolean);

var

NewTime: string[5];

CanEdit: boolean;

CurrentDeparture: T\_Pointer\_Departure;

TimePrompt: string;

begin

CanEdit := True;

CurrentDeparture := Head\_Departure;

while (CurrentDeparture <> nil) and CanEdit do

begin

if CurrentDeparture^.info.MarshrutkaNum = Marshrutka^.info.MarshrutkaNum then

begin

WriteLn('Невозможно изменить время. Маршрутка уже используется в выезде');

CanEdit := False;

end;

CurrentDeparture := CurrentDeparture^.adr;

end;

if CanEdit then

begin

if IsStartTime then

TimePrompt := 'Введите новое время отправления (ЧЧ:ММ): '

else

TimePrompt := 'Введите новое время прибытия (ЧЧ:ММ): ';

GetTime(TimePrompt, NewTime);

if IsStartTime then

Marshrutka^.info.StartTime := NewTime

else

Marshrutka^.info.FinishTime := NewTime;

WriteLn('Время успешно обновлено.');

end;

end;

procedure EditMarshrutkaPrice(Marshrutka: T\_Pointer\_Marshrutka);

var

NewPrice: real;

CanEdit: boolean;

CurrentDeparture: T\_Pointer\_Departure;

begin

CanEdit := True;

CurrentDeparture := Head\_Departure;

while (CurrentDeparture <> nil) and CanEdit do

begin

if CurrentDeparture^.info.MarshrutkaNum = Marshrutka^.info.MarshrutkaNum then

begin

WriteLn('Невозможно изменить цену. Маршрутка уже используется в выезде с кодом: ', CurrentDeparture^.info.DepartureKey);

CanEdit := False;

end;

CurrentDeparture := CurrentDeparture^.adr;

end;

if CanEdit then

begin

WriteLn('Текущая цена: ', Marshrutka^.info.Price:0:2);

GetPrice('Введите новую стоимость проезда', NewPrice);

Marshrutka^.info.Price := NewPrice;

WriteLn('Цена успешно обновлена.');

end;

end;

procedure EditMarshrutkaSeats(Marshrutka: T\_Pointer\_Marshrutka);

var

NewSeats: integer;

CanEdit: boolean;

CurrentDeparture: T\_Pointer\_Departure;

begin

//Проверяем, есть ли маршрутка в выездах

CanEdit := True;

CurrentDeparture := Head\_Departure;

while (CurrentDeparture <> nil) and CanEdit do

begin

if CurrentDeparture^.info.MarshrutkaNum = Marshrutka^.info.MarshrutkaNum then

begin

WriteLn('Невозможно изменить количество мест. Маршрутка уже используется в выезде.');

CanEdit := False;

end;

CurrentDeparture := CurrentDeparture^.adr;

end;

if CanEdit then

begin

WriteLn('Текущее количество мест: ', Marshrutka^.info.Seats);

GetSeats('Введите новое количество мест', NewSeats);

Marshrutka^.info.Seats := NewSeats;

WriteLn('Количество мест успешно обновлено.');

end;

end;

procedure DisplayDepartureInfo(Departure: T\_Pointer\_Departure);

begin

WriteLn('Код выезда: ', Departure^.info.DepartureKey);

WriteLn('Номер маршрутки: ', Departure^.info.MarshrutkaNum);

WriteLn('Дата выезда: ', Departure^.info.Date);

WriteLn('Количество свободных мест: ', Departure^.info.FreeSeats);

end;

procedure EditDepartureDetails(Departure: T\_Pointer\_Departure);

var

Choice: char;

ExitMenu: boolean;

begin

WriteLn;

WriteLn('Выбрано бронирование:');

DisplayDepartureInfo(Departure);

WriteLn;

ExitMenu := False;

while not ExitMenu do

begin

WriteLn('Выберите характеристику для изменения:');

WriteLn('1. Дата отправления');

WriteLn('2. Количество свободных мест');

WriteLn('0. Вернуться в предыдущее меню');

Writeln;

ReadLn(Choice);

case Choice of

'1': EditDepartureDate(Departure);

'2': EditDepartureFreeSeats(Departure);

'0': ExitMenu := True;

else

WriteLn('Неверный выбор. Пожалуйста, выберите снова.');

end;

end;

end;

procedure DisplayTripInfo(Trip: T\_Pointer\_Trip);

var StatusStr: string;

begin

WriteLn('Код бронирования: ', Trip^.info.BookKey);

WriteLn('ФИО пассажира: ', Trip^.info.Name);

WriteLn('Телефон: ', Trip^.info.PhoneNumber);

WriteLn('Количество мест: ', Trip^.info.Seats);

WriteLn('Код выезда: ', Trip^.info.DepartureKey);

case Trip^.info.Status of

bsNotApprove: StatusStr := 'Не подтверждён';

bsApprove: StatusStr := 'Подтверждён';

end;

WriteLn('Статус бронирования: ', StatusStr);

end;

procedure EditTripSeats(Trip: T\_Pointer\_Trip);

var

NewSeatsStr: string;

NewSeats, Code, OldSeats: integer;

ValidInput, CancelOperation: boolean;

CurrentDep: T\_Pointer\_Departure;

DepartureFound: boolean;

Departure: T\_Pointer\_Departure;

begin

WriteLn;

WriteLn('Текущее количество мест: ', Trip^.info.Seats);

OldSeats := Trip^.info.Seats;

CurrentDep := Head\_Departure^.adr;

DepartureFound := False;

Departure := nil;

while (CurrentDep <> nil) and (not DepartureFound) do

begin

if CurrentDep^.info.DepartureKey = Trip^.info.DepartureKey then

begin

DepartureFound := True;

Departure := CurrentDep;

end

else

CurrentDep := CurrentDep^.adr;

end;

if Departure = nil then

begin

WriteLn('Ошибка: Не найден выезд с кодом ', Trip^.info.DepartureKey);

WriteLn('Изменение количества мест невозможно.');

end

else

begin

WriteLn('Доступно свободных мест на выезде: ', Departure^.info.FreeSeats);

ValidInput := False;

CancelOperation := False;

while (not ValidInput) and (not CancelOperation) do

begin

Write('Введите новое количество мест (0 - отмена): ');

ReadLn(NewSeatsStr);

if NewSeatsStr = '0' then

begin

WriteLn('Операция отменена.');

CancelOperation := True;

end

else

begin

Val(NewSeatsStr, NewSeats, Code);

if Code = 0 then

begin

if NewSeats > 0 then

begin

if NewSeats <= (Departure^.info.FreeSeats + OldSeats) then

begin

Departure^.info.FreeSeats := Departure^.info.FreeSeats + OldSeats;

Departure^.info.FreeSeats := Departure^.info.FreeSeats - NewSeats;

Trip^.info.Seats := NewSeats;

ValidInput := True;

WriteLn('Количество мест успешно обновлено.');

end

else

WriteLn('Ошибка: Недостаточно свободных мест. Доступно всего: ', Departure^.info.FreeSeats, ' мест.');

end

else

WriteLn('Ошибка: Количество мест должно быть положительным числом.');

end

else

WriteLn('Ошибка: Введите корректное число.');

end;

end;

end;

end;

procedure EditTripStatus(Trip: T\_Pointer\_Trip);

var

StatusStr, InputInfo: string;

ValidInput: boolean;

begin

WriteLn;

case Trip^.info.status of

bsNotApprove: StatusStr := 'Не подтверждён';

bsApprove: StatusStr := 'Подтверждён';

end;

WriteLn('Текущий статус бронирования: ', StatusStr);

ValidInput := False;

while not ValidInput do

begin

WriteLn('Введите 1 для изменения статуса бронирования, 0 для выхода в предыдущее меню');

ReadLn(InputInfo);

if InputInfo = '1' then

begin

if Trip^.info.status = bsApprove then

Trip^.info.status := bsNotApprove

else

Trip^.info.status := bsApprove;

ValidInput := True;

end

else if InputInfo = '0' then

begin

ValidInput := True;

end

else

WriteLn('Вы ввели некорректное значение. Попробуйте снова.');

end;

end;

procedure DisplayMarshrutkaInfo(Marshrutka: T\_Pointer\_Marshrutka);

begin

WriteLn('Номер маршрутки: ', Marshrutka^.info.MarshrutkaNum);

WriteLn('Место отправления: ', Marshrutka^.info.StartPlace);

WriteLn('Место прибытия: ', Marshrutka^.info.FinishPlace);

WriteLn('Время отправления: ', Marshrutka^.info.StartTime);

WriteLn('Время прибытия: ', Marshrutka^.info.FinishTime);

WriteLn('Цена: ', Marshrutka^.info.Price:0:2, ' руб.');

WriteLn('Вместимость: ', Marshrutka^.info.Seats, ' мест');

end;

procedure EditMarshrutkaDetails(Marshrutka: T\_Pointer\_Marshrutka);

var

Choice: char;

ExitMenu: boolean;

begin

WriteLn;

WriteLn('Выбрана маршрутка:');

DisplayMarshrutkaInfo(Marshrutka);

WriteLn;

ExitMenu := False;

while not ExitMenu do

begin

WriteLn('Выберите характеристику для изменения:');

WriteLn('1. Место отправления');

WriteLn('2. Место прибытия');

WriteLn('3. Время отправления');

WriteLn('4. Время прибытия');

WriteLn('5. Цена');

WriteLn('6. Вместимость маршрутки');

WriteLn('0. Вернуться в предыдущее меню');

Writeln;

ReadLn(Choice);

case Choice of

'1': EditMarshrutkaPlace (Marshrutka, True);

'2': EditMarshrutkaPlace (Marshrutka, False);

'3': EditMarshrutkaTime (Marshrutka, True);

'4': EditMarshrutkaTime (Marshrutka, False);

'5': EditMarshrutkaPrice(Marshrutka);

'6': EditMarshrutkaSeats(Marshrutka);

'0': ExitMenu := True;

else

WriteLn('Неверный выбор. Пожалуйста, выберите снова.');

end;

if not ExitMenu and (Choice in ['1'..'6']) then

begin

WriteLn;

WriteLn('Маршрутка обновлена:');

DisplayMarshrutkaInfo(Marshrutka);

WriteLn;

end;

end;

end;

procedure EditTripDetails(Trip: T\_Pointer\_Trip; Head\_Departure: T\_Pointer\_Departure; Head\_Marshrutka: T\_Pointer\_Marshrutka);

var

Choice: char;

ExitMenu: boolean;

begin

WriteLn;

WriteLn('Выбрано бронирование:');

DisplayTripInfo(Trip);

WriteLn;

ExitMenu := False;

while not ExitMenu do

begin

WriteLn('Выберите характеристику для изменения:');

WriteLn('1. ФИО пассажира');

WriteLn('2. Номер телефона');

WriteLn('3. Количество бронируемых мест');

WriteLn('4. Статус бронирования');

WriteLn('0. Вернуться в предыдущее меню');

Writeln;

ReadLn(Choice);

case Choice of

'1': EditTripName(Trip);

'2': EditTripPhone(Trip);

'3': EditTripSeats(Trip);

'4': EditTripStatus(Trip);

'0': ExitMenu := True;

else

WriteLn('Неверный выбор. Пожалуйста, выберите снова.');

end;

if not ExitMenu and (Choice in ['1'..'4']) then

begin

WriteLn;

WriteLn('Бронирование обновлено:');

DisplayTripInfo(Trip);

WriteLn;

end;

end;

CreateTripInfoFile(Trip^.info, Head\_Departure, Head\_Marshrutka);

end;

procedure ShowMarshrutkaRedactMenu (Head\_Marshrutka: T\_Pointer\_Marshrutka);

var

MarshrutkaNum: string;

Marshrutka: T\_Pointer\_Marshrutka;

Previous: T\_Pointer\_Marshrutka;

Current: T\_Pointer\_Marshrutka;

Found: boolean;

ValidInput: boolean;

begin

if Head\_Marshrutka^.adr = nil then

begin

WriteLn('Список маршруток пуст. Корректировать нечего.');

WriteLn('Нажмите Enter для продолжения...');

ReadLn;

end

else

begin

ViewMarshrutkaList(Head\_Marshrutka);

WriteLn;

ValidInput := False;

Marshrutka := nil;

while not ValidInput do

begin

WriteLn('Введите номер маршрутки для корректировки (0 - отмена):');

ReadLn(MarshrutkaNum);

if MarshrutkaNum = '0' then

ValidInput := True

else

begin

Current := Head\_Marshrutka^.adr;

Previous := Head\_Marshrutka;

Found := False;

while (Current <> nil) and (not Found) do

begin

if Current^.info.MarshrutkaNum = MarshrutkaNum then

Found := True

else

begin

Previous := Current;

Current := Current^.adr;

end;

end;

if Found then

begin

Marshrutka := Current;

ValidInput := True;

end

else

WriteLn('Ошибка: Маршрутка с номером ', MarshrutkaNum, ' не найдена.');

end;

end;

if Marshrutka <> nil then

EditMarshrutkaDetails(Marshrutka);

end;

end;

procedure ShowDepartureRedactMenu(Head\_Departure: T\_Pointer\_Departure);

var

DepartKeyStr: string;

DepartKey: integer;

Code: integer;

ValidInput: boolean;

Departure: T\_Pointer\_Departure;

Previous: T\_Pointer\_Departure;

Current: T\_Pointer\_Departure;

Found: boolean;

Choice: char;

begin

if Head\_Departure^.adr = nil then

begin

WriteLn('Список выездов маршруток пуст. Корректировать нечего.');

WriteLn('Нажмите Enter для продолжения...');

ReadLn;

end

else

begin

ViewDepartureList (Head\_Departure);

WriteLn;

ValidInput := False;

Departure := nil;

while not ValidInput do

begin

WriteLn('Введите код выезда для корректировки (0 - отмена):');

ReadLn(DepartKeyStr);

if DepartKeyStr = '0' then

ValidInput := True

else

begin

Val(DepartKeyStr, DepartKey, Code);

if Code = 0 then

begin

Current := Head\_Departure^.adr;

Previous := Head\_Departure;

Found := False;

while (Current <> nil) and (not Found) do

begin

if Current^.info.DepartureKey = DepartKey then

Found := True

else

begin

Previous := Current;

Current := Current^.adr;

end;

end;

if Found then

begin

Departure := Current;

ValidInput := True;

end

else

WriteLn('Ошибка: Бронирование с кодом ', DepartKey, ' не найдено.');

end

else

WriteLn('Ошибка: Код бронирования должен быть числом.');

end;

end;

if Departure <> nil then

EditDepartureDetails(Departure);

end;

end;

procedure ShowTripRedactMenu (Head\_Trip: T\_Pointer\_Trip);

var

BookKeyStr: string;

BookKey, Code: integer;

ValidInput, Found: boolean;

Previous, Current, Trip: T\_Pointer\_Trip;

Choice: char;

begin

if Head\_Trip^.adr = nil then

begin

WriteLn('Список бронирований пуст. Корректировать нечего.');

WriteLn('Нажмите Enter для продолжения...');

ReadLn;

end

else

begin

ViewTripList (Head\_Trip);

WriteLn;

ValidInput := False;

Trip := nil;

while not ValidInput do

begin

WriteLn('Введите код бронирования для корректировки (0 - отмена):');

ReadLn(BookKeyStr);

if BookKeyStr = '0' then

ValidInput := True

else

begin

Val(BookKeyStr, BookKey, Code);

if Code = 0 then

begin

Current := Head\_Trip^.adr;

Previous := Head\_Trip;

Found := False;

while (Current <> nil) and (not Found) do

begin

if Current^.info.BookKey = BookKey then

Found := True

else

begin

Previous := Current;

Current := Current^.adr;

end;

end;

if Found then

begin

Trip := Current;

ValidInput := True;

end

else

WriteLn('Ошибка: Бронирование с кодом ', BookKey, ' не найдено.');

end

else

WriteLn('Ошибка: Код бронирования должен быть числом.');

end;

end;

if Trip <> nil then

EditTripDetails(Trip, Head\_Departure, Head\_Marshrutka);

end;

end;

procedure HandleInput(var choice: integer);

var

choiceStr: string;

code: integer;

validInput: boolean;

begin

validInput := false;

repeat

ReadLn(choiceStr);

Val(choiceStr, choice, code);

if code <> 0 then

begin

WriteLn('Ошибка: введите числовое значение!');

end

else

validInput := true;

until validInput;

end;

procedure ShowListMenu;

begin

WriteLn ('1. Просмотреть список маршруток ');

Writeln ('2. Просмотреть список выездов маршрутных такси');

WriteLn ('3. Просмотреть список бронирований');

WriteLn ('0. Выход в главное меню');

end;

procedure ViewListMenu (Head\_Marshrutka: T\_Pointer\_Marshrutka; Head\_Departure: T\_Pointer\_Departure; Head\_Trip: T\_Pointer\_Trip);

var choice: integer;

begin

choice := -1;

repeat

ShowListMenu;

HandleInput(choice);

case choice of

1:

begin

ViewMarshrutkaList(Head\_Marshrutka);

end;

2:

begin

ViewDepartureList (Head\_Departure);

end;

3:

begin

ViewTripList(Head\_Trip);

end;

0:

begin

end;

else

WriteLn('Неверный выбор. Попробуйте снова.');

end;

if choice <> 0 then

begin

WriteLn;

WriteLn('Нажмите Enter для продолжения...');

ReadLn;

end;

until choice = 0;

end;

procedure ShowDeleteMenu;

begin

WriteLn ('1. Удалить маршрутку из списка маршруток');

Writeln ('2. Удалить маршрутку из списка выездов');

WriteLn ('3. Удалить бронирование');

Writeln ('4. Найти маршрутку');

WriteLn ('0. Выход в главное меню');

end;

procedure ViewDeleteMenu (Head\_Marshrutka: T\_Pointer\_Marshrutka; Head\_Departure: T\_Pointer\_Departure; Head\_Trip: T\_Pointer\_Trip);

var choice: integer;

begin

choice := -1;

repeat

ShowDeleteMenu;

HandleInput(choice);

case choice of

1:

begin

DeleteMarshrutka (Head\_Marshrutka, Head\_Departure);

end;

2:

begin

DeleteDeparture (Head\_Departure, Head\_Trip);

end;

3:

begin

DeleteTrip (Head\_Trip, Head\_Departure);

end;

4:

begin

SearchMarshrutka (Head\_marshrutka, Head\_Departure, Head\_Trip);

end;

0:

begin

end;

else

WriteLn('Неверный выбор. Попробуйте снова.');

end;

if choice <> 0 then

begin

WriteLn;

WriteLn('Нажмите Enter для продолжения...');

ReadLn;

end;

until choice = 0;

end;

procedure ShowAddMenu;

begin

WriteLn ('1. Добавить новую маршрутку');

Writeln ('2. Добавить новый выезд маршрутки');

WriteLn ('0. Выход в главное меню');

end;

procedure ViewAddMenu (Head\_Marshrutka: T\_Pointer\_Marshrutka; Head\_Departure: T\_Pointer\_Departure);

var choice: integer;

begin

choice := -1;

repeat

ShowAddMenu;

HandleInput(choice);

case choice of

1:

begin

AddMarshrutka (Head\_Marshrutka);

end;

2:

begin

AddDeparture (Head\_Marshrutka, Head\_Departure);

end;

0:

begin

end;

else

WriteLn('Неверный выбор. Попробуйте снова.');

end;

if choice <> 0 then

begin

WriteLn;

WriteLn('Нажмите Enter для продолжения...');

ReadLn;

end;

until choice = 0;

end;

procedure SortMarshrutkaByFinishPlace (Head\_Marshrutka: T\_Pointer\_Marshrutka);

var

Swapped: boolean;

Current, Next: T\_Pointer\_Marshrutka;

TempInfo: T\_Marshrutka;

begin

if (Head\_Marshrutka^.adr <> nil) and (Head\_Marshrutka^.adr^.adr <> nil) then

begin

repeat

Swapped := False;

Current := Head\_Marshrutka^.adr;

while Current^.adr <> nil do

begin

Next := Current^.adr;

if Current^.info.FinishPlace > Next^.info.FinishPlace then

begin

TempInfo := Current^.info;

Current^.info := Next^.info;

Next^.info := TempInfo;

Swapped := True;

end;

Current := Current^.adr;

end;

until not Swapped;

WriteLn('Маршрутки отсортированы по месту прибытия.');

end

else

begin

WriteLn('Список маршруток пуст или содержит только один элемент. Сортировка не требуется.');

end;

ViewMarshrutkaList(Head\_Marshrutka);

end;

function CompareDates(Date1, Date2: string): boolean;

var

Day1, Month1, Day2, Month2: integer;

Code: integer;

begin

Val(Copy(Date1, 1, 2), Day1, Code);

Val(Copy(Date1, 4, 2), Month1, Code);

Val(Copy(Date2, 1, 2), Day2, Code);

Val(Copy(Date2, 4, 2), Month2, Code);

if Month1 > Month2 then

Result := True

else if Month1 < Month2 then

Result := False

else

Result := Day1 > Day2;

end;

procedure SortDeparturesByDate (Head\_Departure: T\_Pointer\_Departure);

var

Swapped: boolean;

Current, Next: T\_Pointer\_Departure;

TempInfo: T\_Departure;

begin

if (Head\_Departure^.adr <> nil) and (Head\_Departure^.adr^.adr <> nil) then

begin

repeat

Swapped := False;

Current := Head\_Departure^.adr;

while Current^.adr <> nil do

begin

Next := Current^.adr;

if CompareDates(Current^.info.Date, Next^.info.Date) then

begin

TempInfo := Current^.info;

Current^.info := Next^.info;

Next^.info := TempInfo;

Swapped := True;

end;

Current := Current^.adr;

end;

until not Swapped;

WriteLn('Выезды отсортированы по дате.');

end

else

begin

WriteLn('Список выездов пуст или содержит только один элемент. Сортировка не требуется.');

end;

ViewDepartureList (Head\_Departure);

end;

procedure SortTripsByDepartureKey (Head\_Trip: T\_Pointer\_Trip);

var

Swapped: boolean;

Current, Next: T\_Pointer\_Trip;

TempInfo: T\_Trip;

begin

if (Head\_Trip^.adr <> nil) and (Head\_Trip^.adr^.adr <> nil) then

begin

repeat

Swapped := False;

Current := Head\_Trip^.adr;

while Current^.adr <> nil do

begin

Next := Current^.adr;

if Current^.info.DepartureKey > Next^.info.DepartureKey then

begin

TempInfo := Current^.info;

Current^.info := Next^.info;

Next^.info := TempInfo;

Swapped := True;

end;

Current := Current^.adr;

end;

until not Swapped;

WriteLn('Бронирования отсортированы по коду выезда.');

end

else

begin

WriteLn('Список бронирований пуст или содержит только один элемент. Сортировка не требуется.');

end;

ViewTripList(Head\_Trip);

end;

procedure ShowRedactMenu;

begin

WriteLn ('1. Изменить информацию о маршрутке');

Writeln ('2. Изменить информацию о выезде маршрутки');

Writeln ('3. Изменить информацию о бронировании');

Writeln ('4. Найти маршрутку');

WriteLn ('0. Выход в главное меню');

end;

procedure ViewRedactMenu (Head\_Marshrutka: T\_Pointer\_Marshrutka; Head\_Departure: T\_Pointer\_Departure; Head\_Trip: T\_Pointer\_Trip);

var choice: integer;

begin

choice := -1;

repeat

ShowRedactMenu;

HandleInput(choice);

case choice of

1:

begin

ShowMarshrutkaRedactMenu (Head\_Marshrutka);

end;

2:

begin

ShowDepartureRedactMenu (Head\_Departure);

end;

3:

begin

ShowTripRedactMenu (Head\_Trip);

end;

4:

begin

SearchMarshrutka (Head\_marshrutka, Head\_Departure, Head\_Trip);

end;

0:

begin

end;

else

WriteLn('Неверный выбор. Попробуйте снова.');

end;

if choice <> 0 then

begin

WriteLn;

WriteLn('Нажмите Enter для продолжения...');

ReadLn;

end;

until choice = 0;

end;

procedure ShowSortMenu;

begin

WriteLn ('1. Сортировать маршрутки по месту прибытия');

WriteLn ('2. Сортировать выезды по дате');

WriteLn ('3. Сортировать бронирования по коду выезда');

WriteLn ('0. Выход в главное меню');

end;

procedure ViewSortMenu;

var choice: integer;

begin

choice := -1;

repeat

ShowSortMenu;

HandleInput(choice);

case choice of

1:

begin

SortMarshrutkaByFinishPlace (Head\_Marshrutka);

end;

2:

begin

SortDeparturesByDate (Head\_Departure);

end;

3:

begin

SortTripsByDepartureKey (Head\_Trip);

end;

0:

begin

end;

else

WriteLn('Неверный выбор. Попробуйте снова.');

end;

if choice <> 0 then

begin

WriteLn;

WriteLn('Нажмите Enter для продолжения...');

ReadLn;

end;

until choice = 0;

end;

procedure ShowMenu;

begin

WriteLn ('===== УПРАВЛЕНИЕ МАРШРУТКАМИ =====');

WriteLn ('1. Чтение данных из файла');

WriteLn ('2. Просмотр списка');

WriteLn ('3. Сортировка элементов списка');

WriteLn ('4. Поиск маршрутки');

WriteLn ('5. Добавление элементов списка');

WriteLn ('6. Удаление элементов списка');

Writeln ('7. Изменение элементов списка');

Writeln ('8. Забронировать место в маршрутке');

WriteLn ('9. Выход без сохранения');

WriteLn ('10. Выход с сохранением');

WriteLn ('================================');

end;

procedure LoadMarshrutkaFile(var Head: T\_Pointer\_Marshrutka);

var

MarshrutkaFile: file of T\_Marshrutka;

TempMarshrutka: T\_Marshrutka;

NewNode, Current: T\_Pointer\_Marshrutka;

FileExists: boolean;

begin

AssignFile(MarshrutkaFile, 'marshrutka.dat');

Reset(MarshrutkaFile);

while not Eof(MarshrutkaFile) do

begin

Read(MarshrutkaFile, TempMarshrutka);

New(NewNode);

NewNode^.Info := TempMarshrutka;

NewNode^.adr := nil;

if Head = nil then

Head := NewNode

else

begin

Current := Head;

while Current^.adr <> nil do

Current := Current^.adr;

Current^.adr := NewNode;

end;

end;

CloseFile(MarshrutkaFile);

end;

procedure LoadDepartureFile(var Head: T\_Pointer\_Departure);

var

DepartureFile: file of T\_Departure;

TempDeparture: T\_Departure;

NewNode, Current: T\_Pointer\_Departure;

begin

AssignFile(DepartureFile, 'departure.dat');

Reset(DepartureFile);

while not Eof(DepartureFile) do

begin

Read(DepartureFile, TempDeparture);

if TempDeparture.DepartureKey > LastDepartureKey then

LastDepartureKey := TempDeparture.DepartureKey;

New(NewNode);

NewNode^.Info := TempDeparture;

NewNode^.adr := nil;

if Head = nil then

Head := NewNode

else

begin

Current := Head;

while Current^.adr <> nil do

Current := Current^.adr;

Current^.adr := NewNode;

end;

end;

CloseFile(DepartureFile);

end;

procedure LoadTripFile(var Head: T\_Pointer\_Trip);

var

TripFile: file of T\_Trip;

TempTrip: T\_Trip;

NewNode, Current: T\_Pointer\_Trip;

begin

AssignFile(TripFile, 'trip.dat');

Reset(TripFile);

while not Eof(TripFile) do

begin

Read(TripFile, TempTrip);

if TempTrip.BookKey > LastBookKey then

LastBookKey := TempTrip.BookKey;

New(NewNode);

NewNode^.Info := TempTrip;

NewNode^.adr := nil;

if Head = nil then

Head := NewNode

else

begin

Current := Head;

while Current^.adr <> nil do

Current := Current^.adr;

Current^.adr := NewNode;

end;

end;

CloseFile(TripFile);

end;

procedure LoadAllFiles (Ьead\_Marshrutka: T\_Pointer\_Marshrutka; Head\_Departure: T\_Pointer\_Departure; Head\_Trip: T\_Pointer\_Trip);

begin

LoadMarshrutkaFile(Head\_Marshrutka);

LoadDepartureFile(Head\_Departure);

LoadTripFile(Head\_Trip);

writeln ('Загрузка данных завершена');

end;

procedure DisposeAllLists(Head\_Marshrutka: T\_Pointer\_Marshrutka; Head\_Departure: T\_Pointer\_Departure; Head\_Trip: T\_Pointer\_Trip);

var

CurrentMarshrutka, TempMarshrutka: T\_Pointer\_Marshrutka;

CurrentDeparture, TempDeparture: T\_Pointer\_Departure;

CurrentTrip, TempTrip: T\_Pointer\_Trip;

begin

CurrentMarshrutka := Head\_Marshrutka^.adr;

while CurrentMarshrutka <> nil do

begin

TempMarshrutka := CurrentMarshrutka;

CurrentMarshrutka := CurrentMarshrutka^.adr;

Dispose(TempMarshrutka);

end;

Dispose(Head\_Marshrutka);

CurrentDeparture := Head\_Departure^.adr;

while CurrentDeparture <> nil do

begin

TempDeparture := CurrentDeparture;

CurrentDeparture := CurrentDeparture^.adr;

Dispose(TempDeparture);

end;

Dispose(Head\_Departure);

CurrentTrip := Head\_Trip^.adr;

while CurrentTrip <> nil do

begin

TempTrip := CurrentTrip;

CurrentTrip := CurrentTrip^.adr;

Dispose(TempTrip);

end;

Dispose(Head\_Trip);

WriteLn('Память успешно освобождена.');

end;

begin

New(Head\_Marshrutka);

New(Head\_Departure);

New(Head\_Trip);

Head\_Marshrutka^.adr := nil;

Head\_Departure^.adr := nil;

Head\_Trip^.adr := nil;

LastDepartureKey := 0;

LastBookKey := 0;

isFirstAction := true;

choice := -1;

repeat

ShowMenu;

HandleInput(choice);

case choice of

1:

begin

if isFirstAction then

begin

LoadAllFiles(Head\_Marshrutka, Head\_Departure, Head\_Trip);

isFirstAction := false;

end

else

begin

WriteLn('Загружать данные можно только в начале работы программы!');

WriteLn('Нажмите Enter для продолжения...');

ReadLn;

end;

end;

2:

begin

ViewListMenu(Head\_Marshrutka, Head\_Departure, Head\_Trip);

isFirstAction := false;

end;

3:

begin

ViewSortMenu;

isFirstAction := false;

end;

4:

begin

SearchMarshrutka (Head\_marshrutka, Head\_Departure, Head\_Trip);

isFirstAction := false;

end;

5:

begin

ViewAddMenu (Head\_Marshrutka, Head\_Departure);

isFirstAction := false;

end;

6:

begin

ViewDeleteMenu (Head\_Marshrutka, Head\_Departure, Head\_Trip);

isFirstAction := false;

end;

7:

begin

ViewRedactMenu (Head\_Marshrutka, Head\_Departure, Head\_Trip);

isFirstAction := false;

end;

8:

begin

AddTrip (Head\_Marshrutka, Head\_Departure, Head\_Trip);

isFirstAction := false;

end;

9:

begin

isFirstAction := false;

DisposeAllLists (Head\_Marshrutka, Head\_Departure, Head\_Trip);

end;

10:

begin

SaveDataToFiles (Head\_Marshrutka, Head\_Departure, Head\_Trip);

isFirstAction := false;

DisposeAllLists (Head\_Marshrutka, Head\_Departure, Head\_Trip);

end;

else

WriteLn('Неверный выбор. Попробуйте снова.');

end;

WriteLn;

WriteLn('Нажмите Enter для продолжения...');

ReadLn;

until (choice = 9) or (choice = 10);

end.