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 16](#_Toc199713670)

[3.1 Разработка схемы программы 16](#_Toc199713671)

[3.2. Выделение основных структур данных 28](#_Toc199713672)

[3.3. Описание структур в виде таблиц 29](#_Toc199713673)

[3.4. Определение подпрограмм и их описание 33](#_Toc199713674)

[3.5 Тестирование и отладка программы 40](#_Toc199713675)

[4. РУКОВОДСТВО ПО УСТАНОВКЕ И ЭКСПЛУАТАЦИИ 40](#_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 44](#_Toc199713680)

[4.2.3. Форматы ввода данных 45](#_Toc199713681)

[4.2.4. Работа с файлами 45](#_Toc199713682)

[4.3. Возможные проблемы и их решения 46](#_Toc199713683)

[Приложение 1. Код программы 47](#_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 блока инициализации. На каждой итерации цикла пользователю предоставляется меню, после чего программа обрабатывает сделанный выбор и выполняет соответствующее действие, вызывая нужную подсистему.

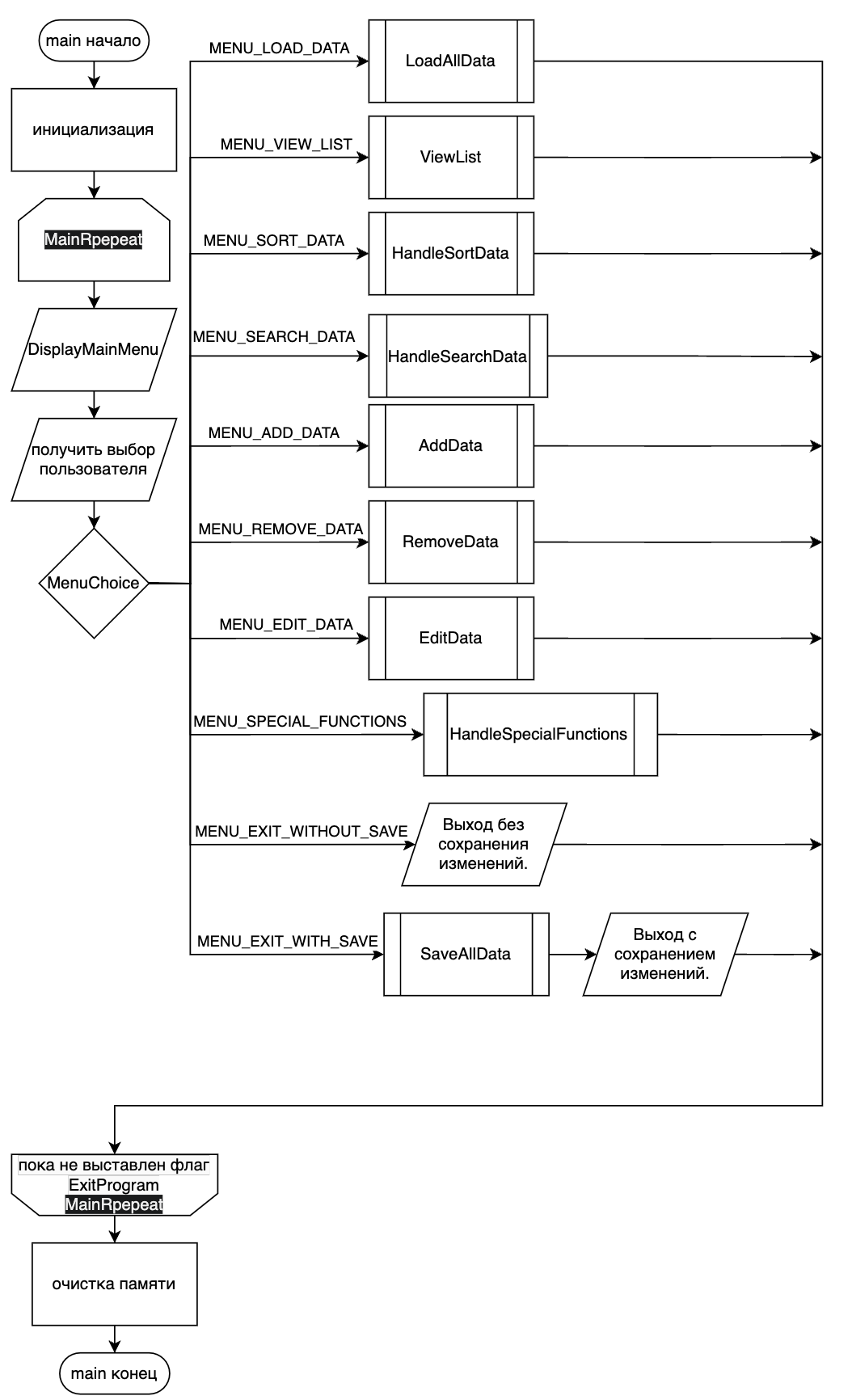


Рисунок 3.1 – Схема работы основного меню программы

Данная схема детализирует работу функции ViewList, которая предоставляет пользователю подменю для выбора одного из списков для просмотра: типы компонентов, компоненты, совместимость или заказы. В зависимости от выбора вызывается соответствующая функция отображения. Каждая из этих функций (DisplayComponentTypes, DisplayComponents и т.д.) реализует унифицированный алгоритм вывода данных: очистка экрана, печать заголовка и поэлементный вывод списка в табличном виде.

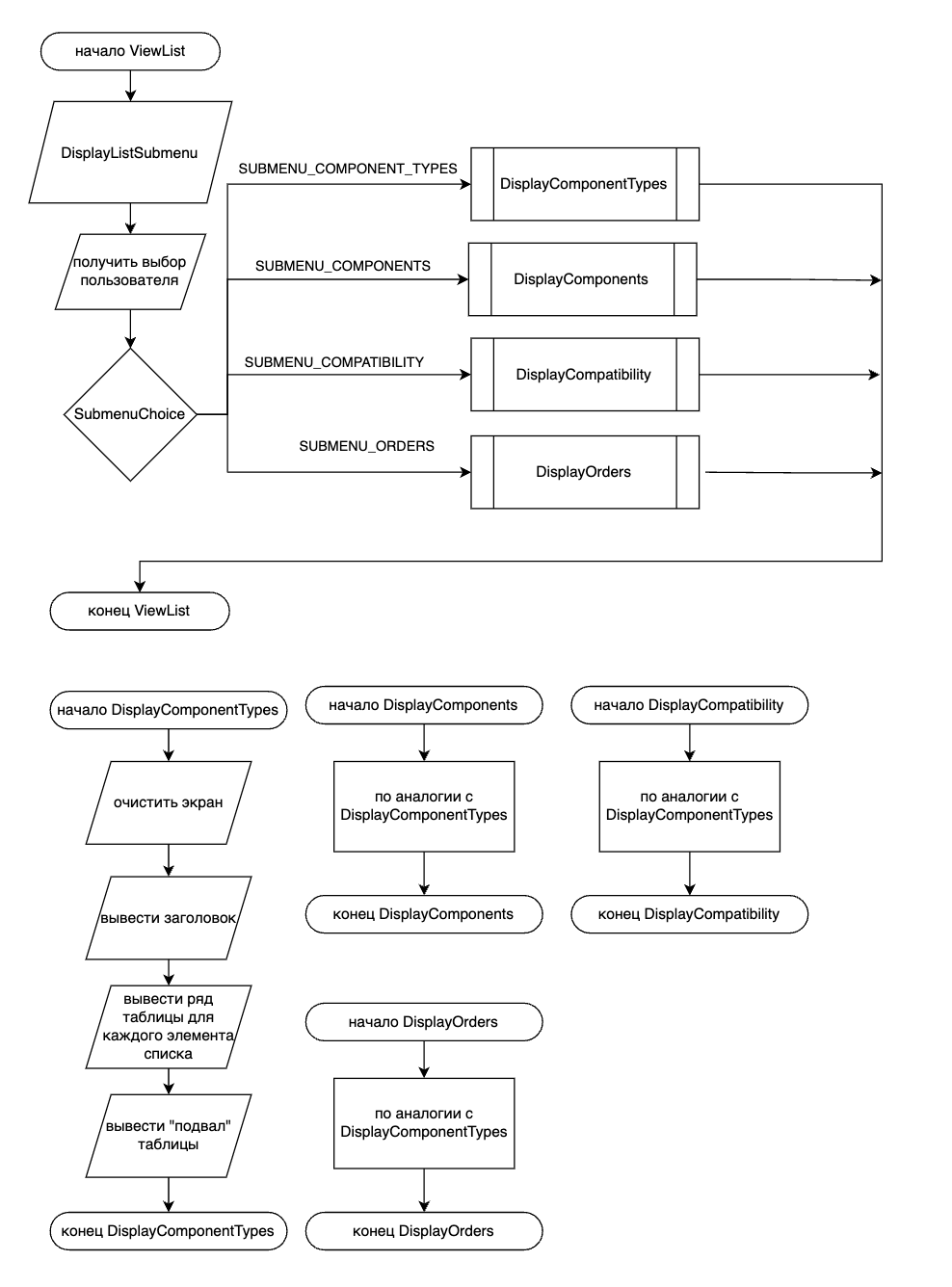


Рисунок 3.2 – Схема работы процедур просмотра списков

Данная схема описывает алгоритм функции LoadAllData, отвечающей за загрузку данных из файлов. Перед выполнением операции система запрашивает у пользователя подтверждение на замену текущих данных в памяти. В случае положительного ответа происходит последовательная загрузка списков типов компонентов, самих компонентов, правил их совместимости и заказов. Если соответствующий файл не найден, создается пустой. По завершении операции пользователь информируется об успехе или отмене действия.

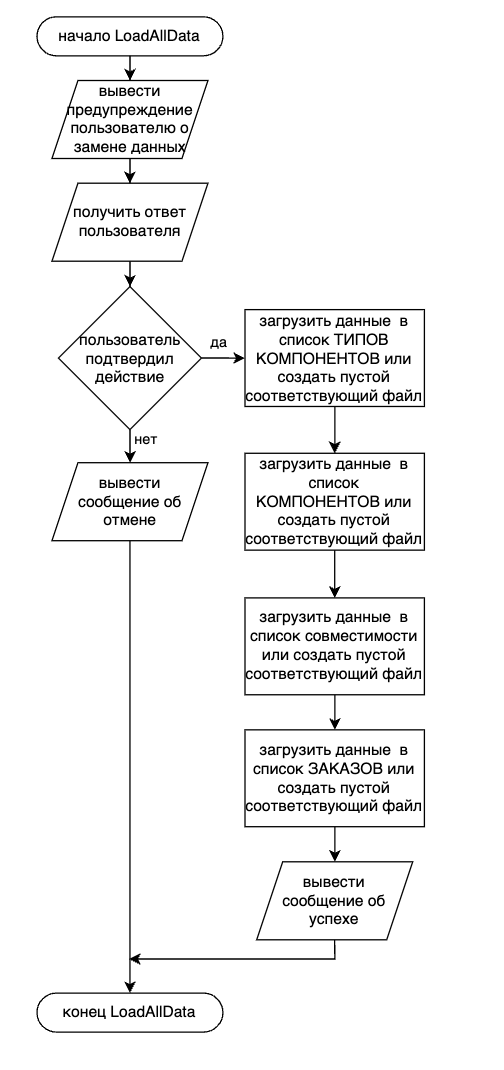


Рисунок 3.3 – Схема работы процедуры загрузки данных из файлов

Данная схема иллюстрирует работу функции-диспетчера HandleSortData. Она выводит пользователю меню с вариантами сортировки данных (типов по коду, компонентов по цене и т.д.) и, в зависимости от выбора, вызывает соответствующую процедуру. В качестве примера детализирован алгоритм SortComponentsByCode, реализующий сортировку пузырьком; остальные функции сортировки построены по аналогии. После выполнения операции система информирует пользователя об успехе.

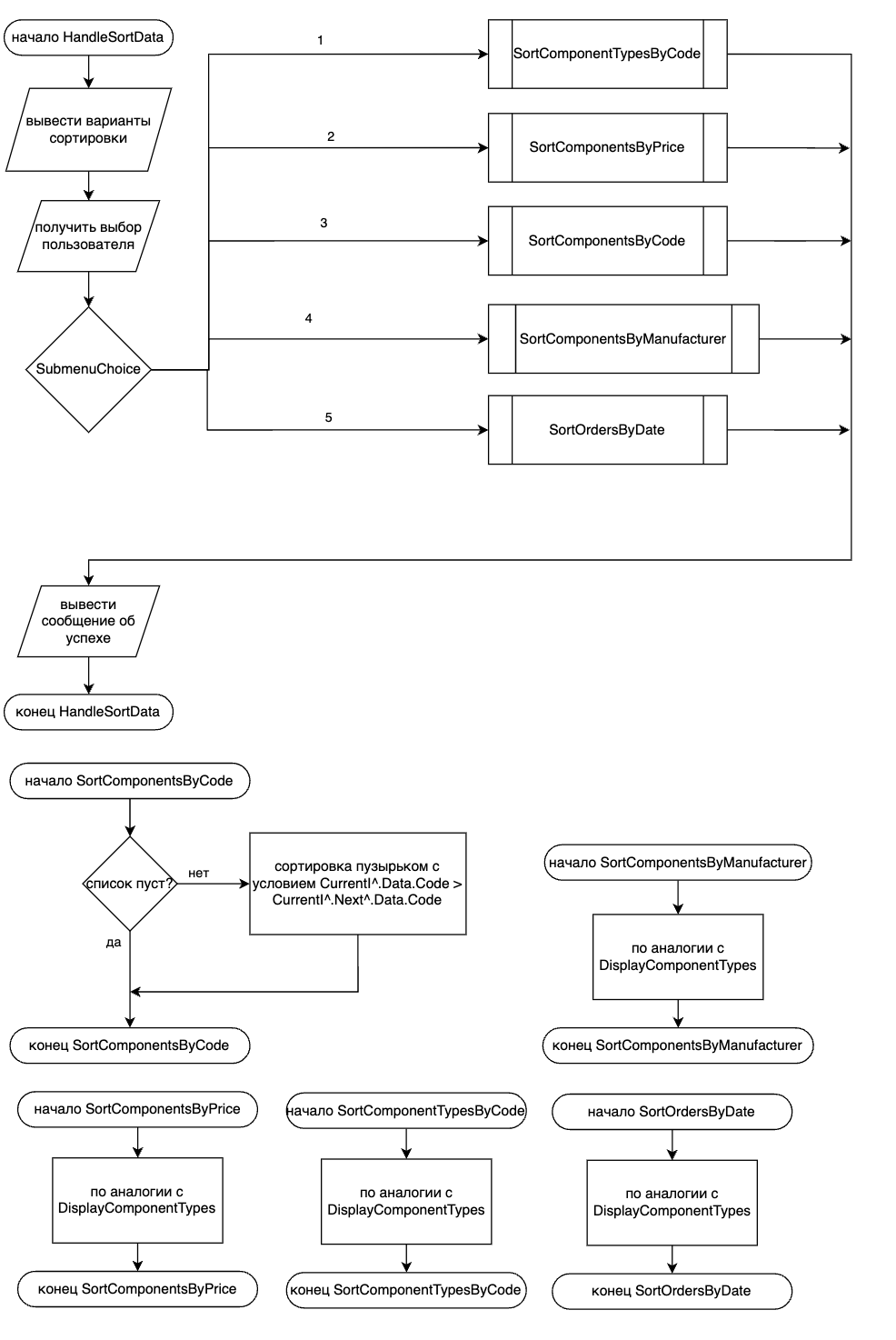


Рисунок 3.4. – Схема работы подсистемы сортировки данных

Данная схема описывает алгоритм работы функции-диспетчера HandleSearchData. Функция предлагает пользователю меню с вариантами поиска (по диапазону цен, производителю, типу и т.д.), получает выбор и необходимые для поиска параметры. В зависимости от выбора вызывается одна из специализированных функций поиска. Логика работы этих функций унифицирована и показана на примере SearchComponentsByType: система последовательно перебирает все элементы основного списка, и если элемент удовлетворяет условию поиска, он добавляется в отдельный список результатов.

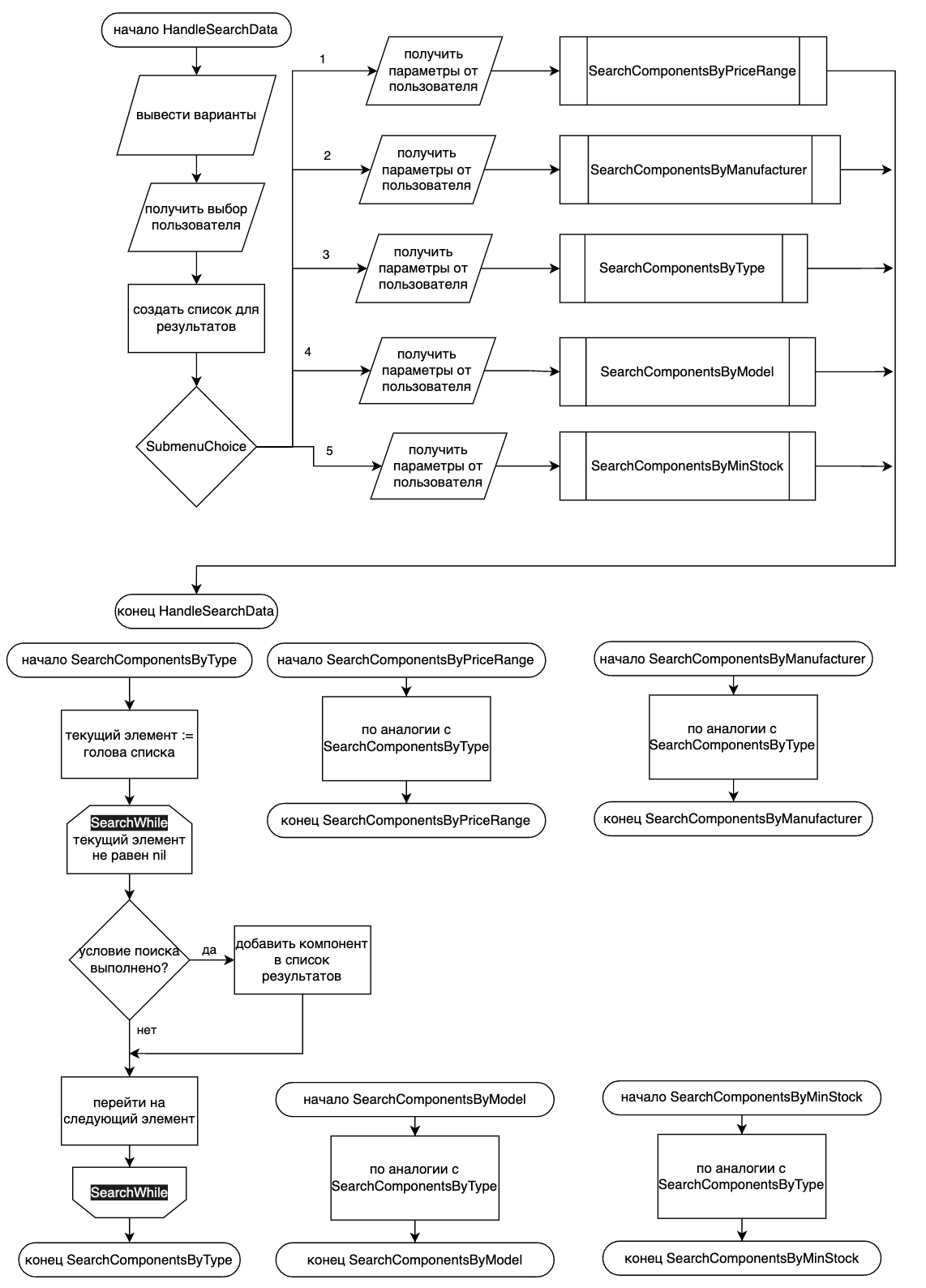


Рисунок 3.5. – Схема работы процедур поиска

Схема иллюстрирует алгоритм добавления данных `AddData`. На основе выбора пользователя и после предварительных проверок целостности, функция-диспетчер вызывает подпрограмму для добавления типа, компонента или правила совместимости. Также показана логика процедур ввода, включающая получение данных, их валидацию и обработку ошибок.

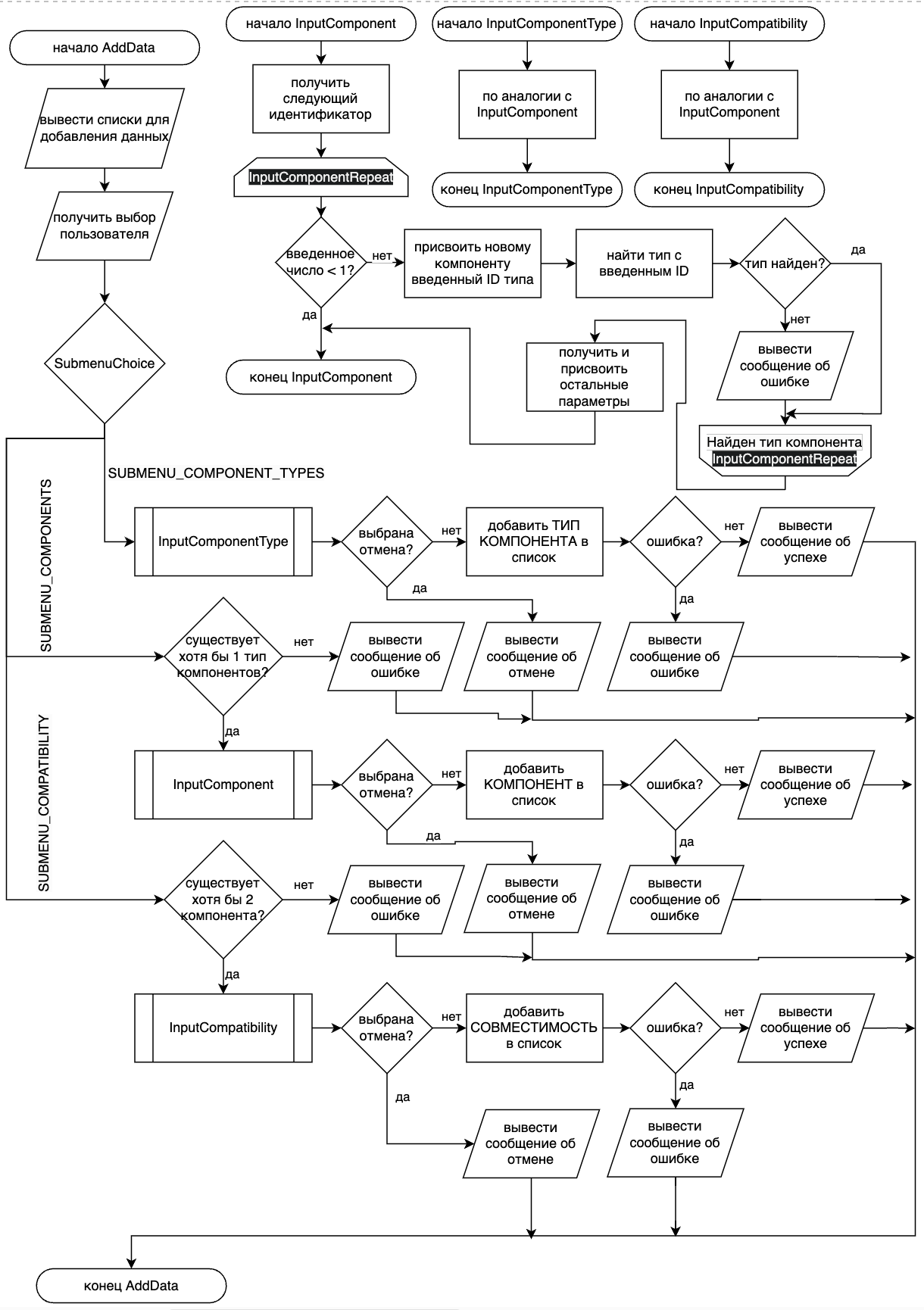


Рисунок 3.6. – Схема работы процедур добавления данных

Схема описывает диспетчер `RemoveData`, который по выбору пользователя инициирует удаление типа, компонента или правила совместимости. Алгоритм, детализированный на примере `RemoveComponentType`, включает ключевой шаг: проверку зависимостей. Перед удалением типа система сканирует список компонентов. Удаление записи производится только при отсутствии зависимых объектов или после получения подтверждения от пользователя.

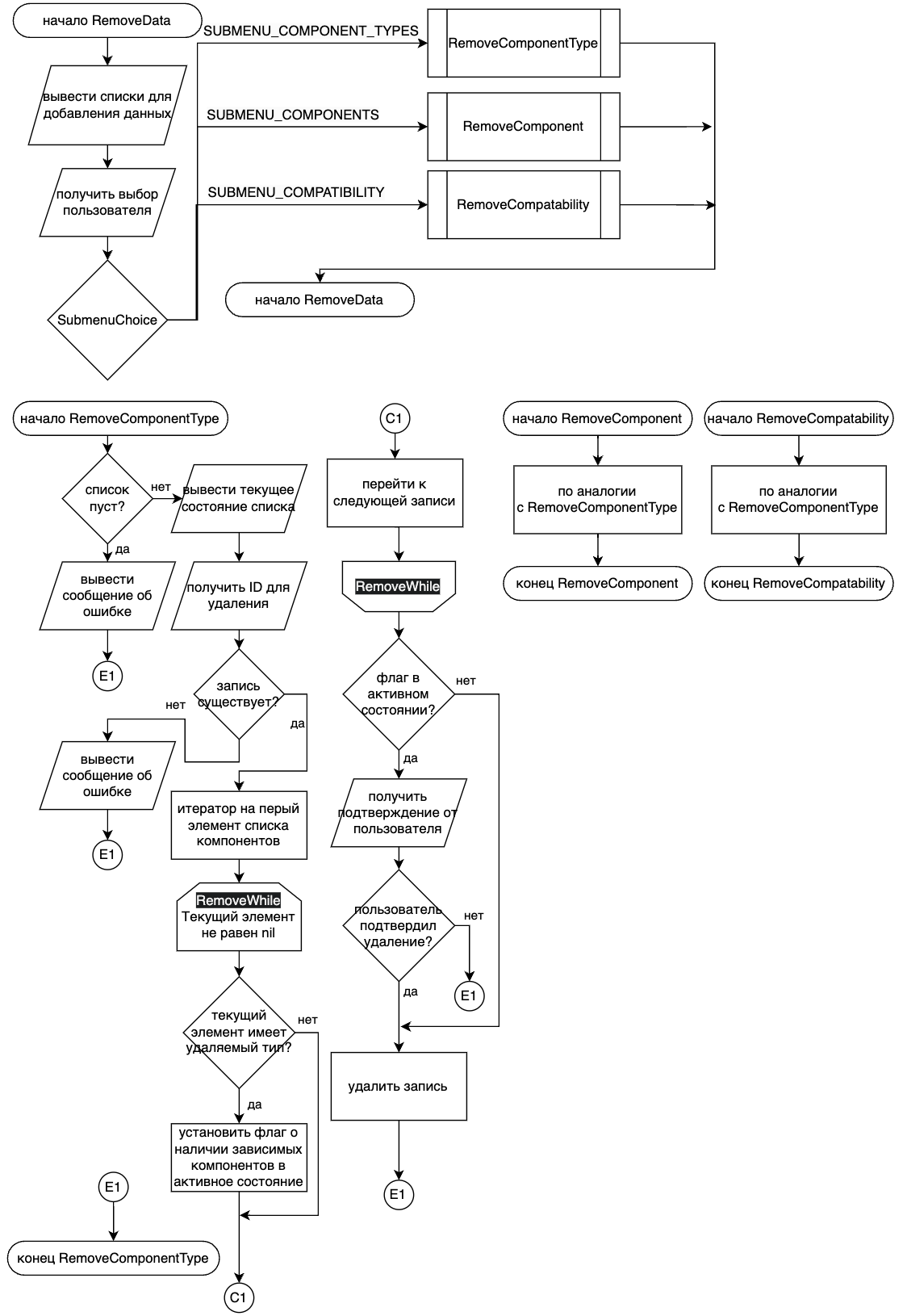


Рисунок 3.7. – Схема работы процедур удаления

Схема описывает диспетчер `EditData`, вызывающий подпрограммы редактирования для типов, компонентов или правил совместимости. Алгоритм, показанный на примере `EditComponent`, включает выбор записи по ID, ввод новых данных и их последующую валидацию. Ключевой шаг — проверка существования зависимых записей, например, нового указываемого ID типа, перед окончательным обновлением данных.

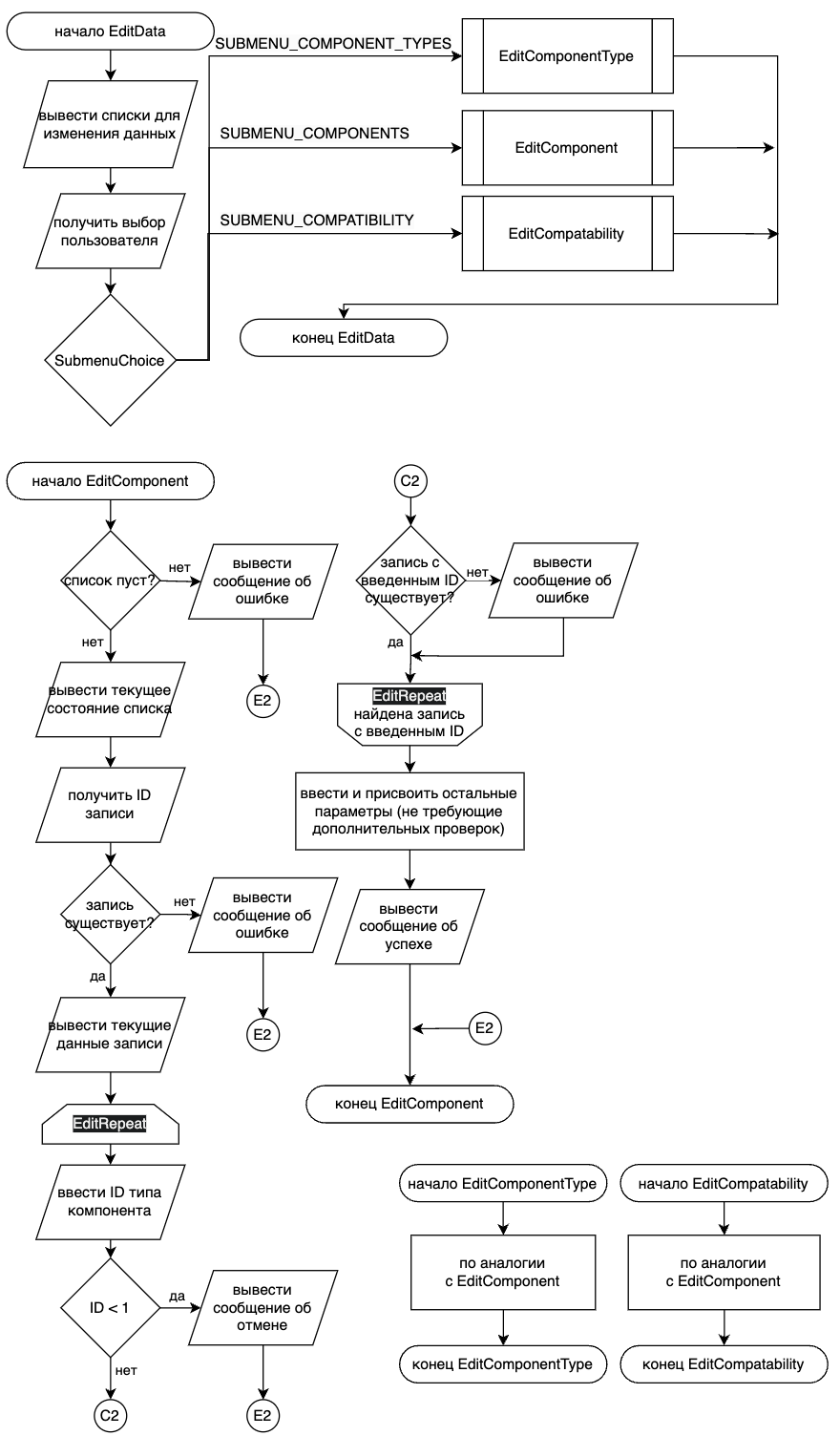


Рисунок 3.8. – Схема работы процедур изменения данных

Схема описывает диспетчер `HandleSpecialFunctions`, который управляет основными функциями программы. В зависимости от выбора пользователя, выполняются три ключевые операции: подбор всех вариантов сборки ПК в заданном ценовом диапазоне (`FindPCBuildOptions`), оформление заказа на одну из найденных сборок (`CreateOrder`) и поиск совместимых комплектующих для конкретного компонента (`FindCompatibleComponents`). Алгоритм включает предварительные проверки на наличие данных, необходимых для каждой операции.

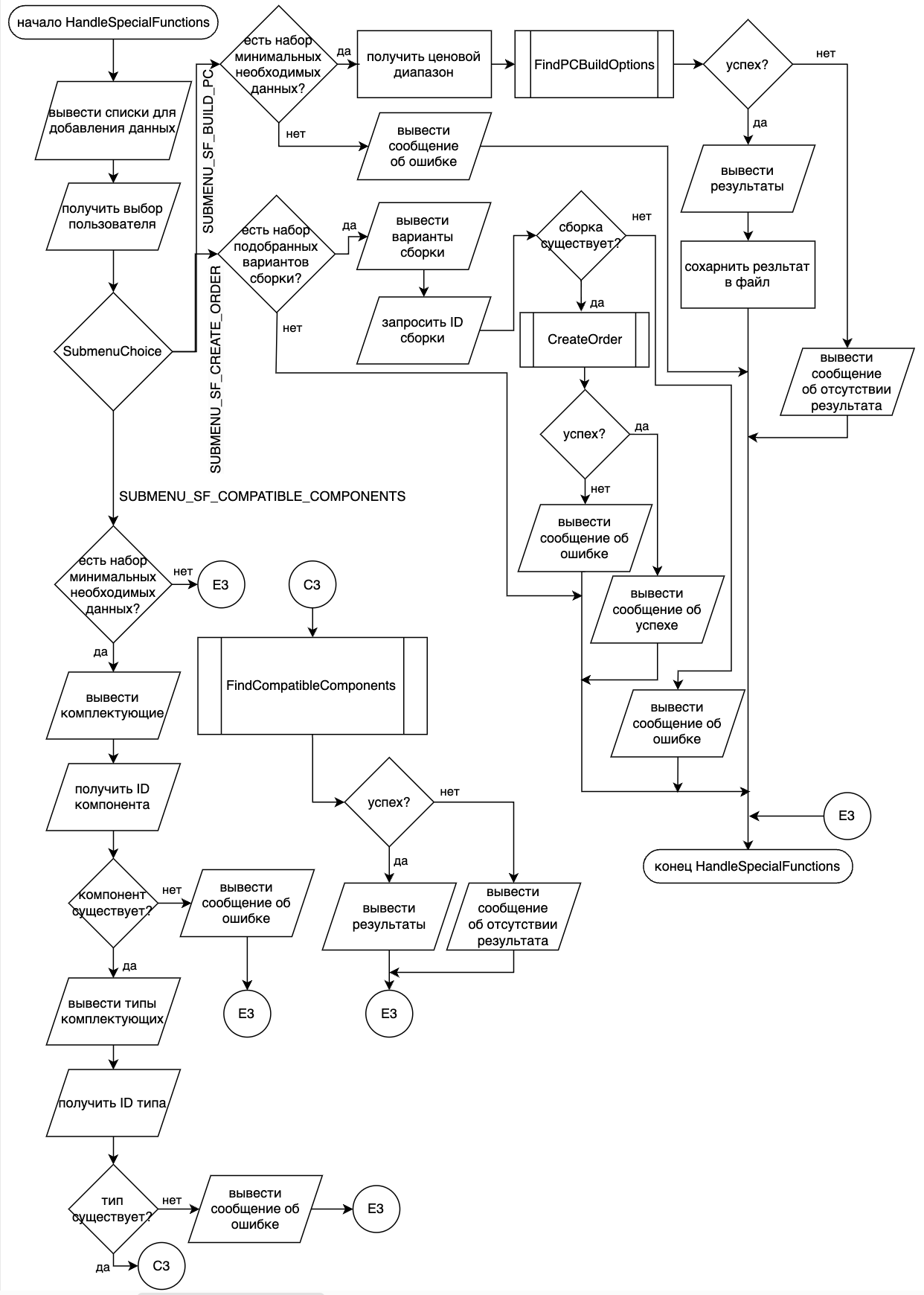


Рисунок 3.9. – Схема работы процедуры-диспатчера для специальных функций.

Схема детализирует алгоритм рекурсивного подбора конфигураций. `FindPCBuildOptions` является функцией-оберткой, инициализирующей процесс. Основная логика заключена в `BuildConfigurationsRecursive`, которая последовательно перебирает типы комплектующих. Для каждого кандидата на добавление в сборку вызывается `IsCompatibleWithAll`. Эта функция, в свою очередь, в цикле обращается к `AreComponentsCompatible`, чтобы проверить совместимость нового компонента с каждым уже добавленным. `AreComponentsCompatible` выполняет прямой поиск по глобальному списку совместимости. Готовая сборка сохраняется, если ее цена соответствует заданному диапазону.

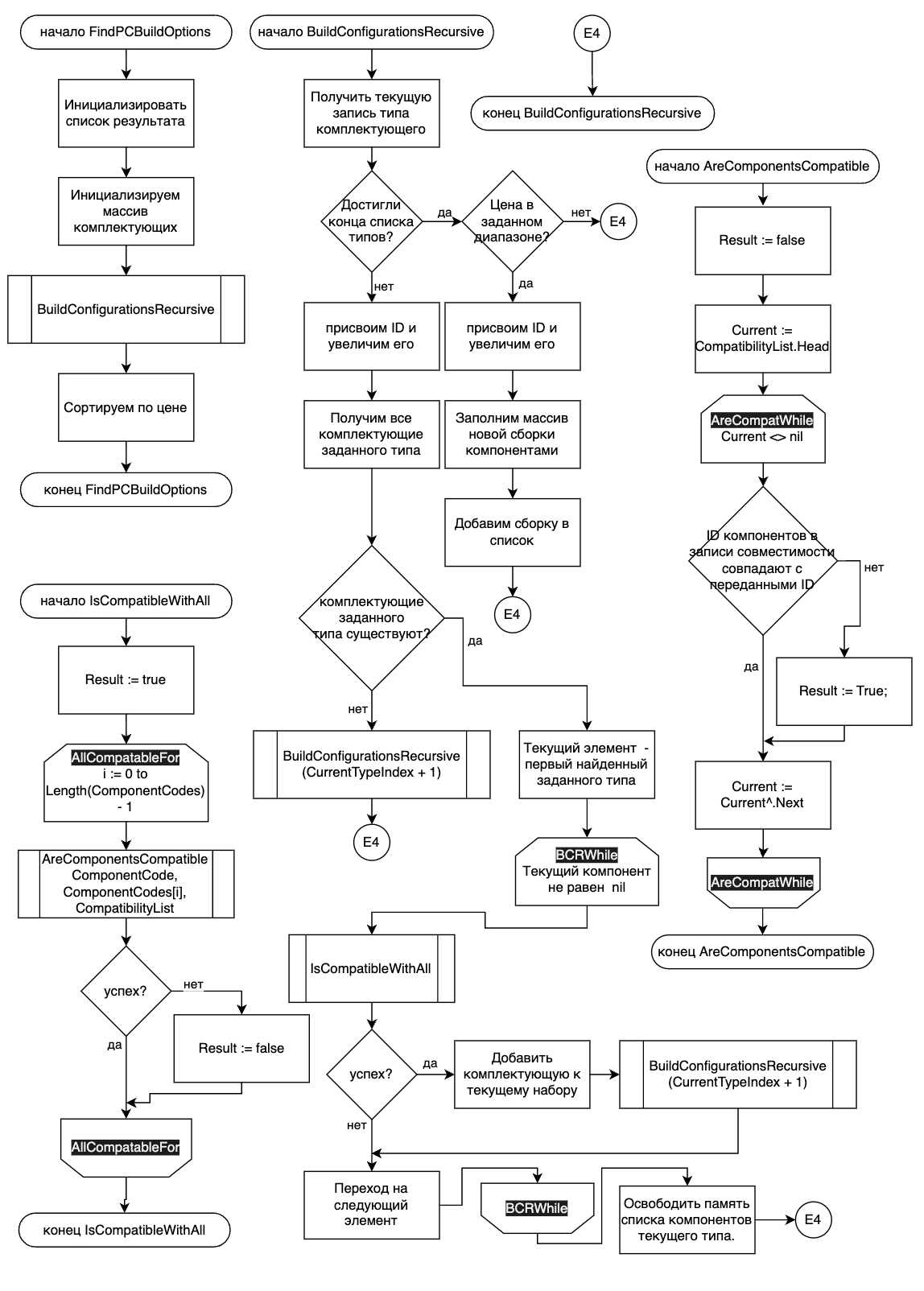


Рисунок 3.10. – Схема работы процедур подбора сборок и  
проверок совместимости

Схема представляет два независимых алгоритма: поиск совместимых комплектующих и создание заказа.

Алгоритм FindCompatibleComponents находит для указанного базового компонента все совместимые детали заданного типа. Он итерирует по полному списку комплектующих, используя функцию AreComponentsCompatible для проверки совместимости. Подходящие компоненты добавляются в результирующий список.

Алгоритм CreateOrder формирует заказ на основе выбранной сборки. Он запрашивает данные заказчика, затем в цикле обрабатывает каждый компонент сборки, проверяя его наличие на складе и уменьшая количество. Готовый заказ добавляется в общий список и сохраняется в отдельный текстовый файл.

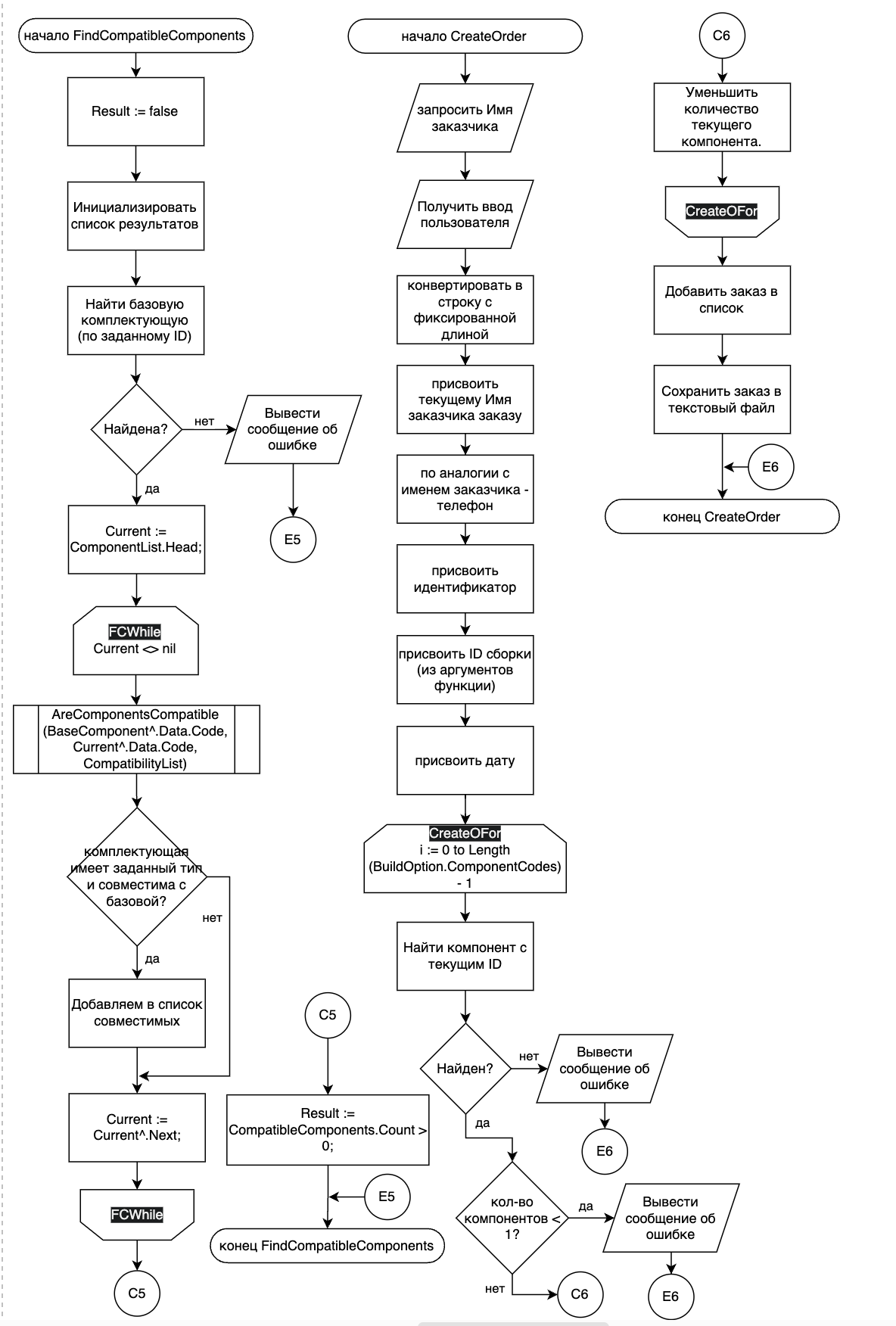


Рисунок 3.11. – Схема работы процедур поиска совместимых компонентов и создания заказа

Схема описывает функцию-диспетчер `SaveAllData`, которая выполняет последовательное сохранение всех данных: типов, компонентов, правил совместимости и заказов. Процесс прерывается при первой же ошибке. Детально показан алгоритм для `SaveComponentTypes`, который в цикле обходит список, преобразует каждую запись в надлежащий для записи формат и записывает ее в файл. Остальные функции сохранения работают по аналогичному принципу.

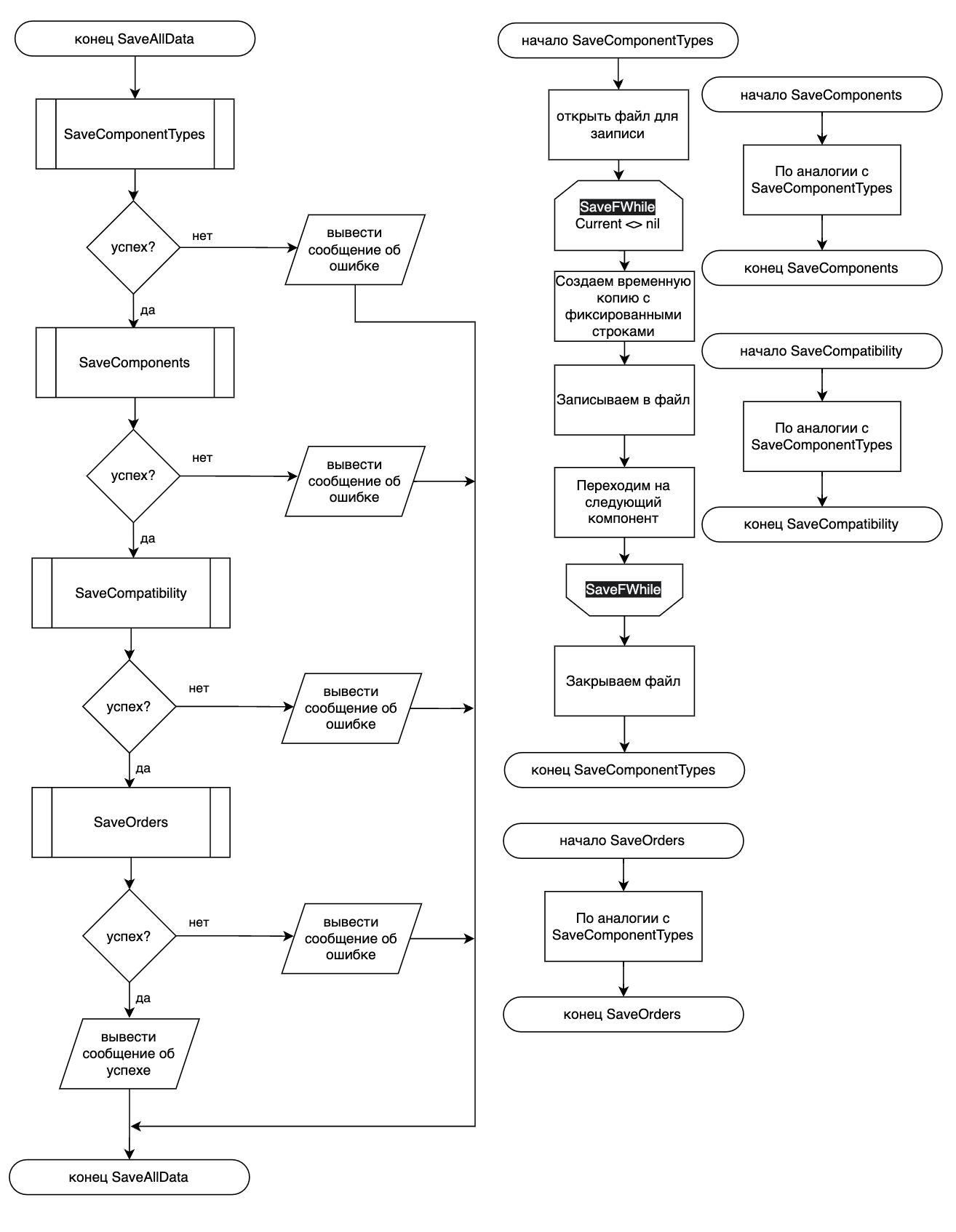


Рисунок 3.12. – Схема работы процедуры сохранения данных

### 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 | Запись (record) |
| TCompatibility | Хранение информации о совместимости комплектующих | Запись (record) |
| TPCBuildOption | Хранение информации о вариантах комплектации ПК | Запись (record) |
| TOrder | Хранение информации о заказах | Запись (record) |
| TComponentTypeNode | Узел односвязного списка типов комплектующих | Запись (record) |
| TComponentNode | Узел односвязного списка комплектующих | Запись (record) |

|  |  |  |
| --- | --- | --- |
| Имя идентификатора структуры | Назначение структуры | Тип структуры |
| TCompatibilityNode | Узел односвязного списка совместимости | Запись (record) |
| TPCBuildOptionNode | Узел односвязного списка вариантов комплектации | Запись (record) |
| TOrderNode | Узел односвязного списка заказов | Запись (record) |
| PComponentTypeNode | Указатель на узел списка типов комплектующих | Указатель |
| PComponentNode | Указатель на узел списка комплектующих | Указатель |
| PCompatibilityNode | Указатель на узел списка совместимости | Указатель |
| PPCBuildOptionNode | Указатель на узел списка вариантов комплектации | Указатель | |
| POrderNode | Указатель на узел списка заказов | Указатель |
| TComponentTypeList | Список типов комплектующих | Запись |
| TComponentList | Список комплектующих | Запись |
| TCompatibilityList | Список правил совместимости | Запись |
| TPCBuildOptionList | Список вариантов комплектации ПК | Запись |
| TOrderList | Список заказов | Запись |

Таблица 3.1 – продолжение

### 

### 3.3. Описание структур в виде таблиц

Таблица 3.2 Используемые идентификаторы

| Таблица 3.2 – Продолжение | | | |
| --- | --- | --- | --- |
| Имя идентификатора | Тип данных | Назначение | Структура |
| TypeCode | Integer | Код типа комплектующей | TComponentType |
| Name | TFixedString | Название типа комплектующей | TComponentType |
| Code | Integer | Уникальный код комплектующей | TComponent |
| TypeCode | Integer | Код типа комплектующей | TComponent |
| Manufacturer | TFixedString | Фирма-изготовитель | TComponent |
| Model | TFixedString | Модель комплектующей | TComponent |
| Parameters | TFixedString | Параметры и характеристики | TComponent |
| Price | Real | Цена комплектующей | TComponent |
| InStock | Integer | Количество в наличии | TComponent |
| ComponentCode1 | Integer | Код первой комплектующей | TCompatibility |
| ComponentCode2 | Integer | Код второй совместимой комплектующей | TCompatibility |
| ID | Integer | Уникальный идентификатор варианта | TPCBuildOption |
| ComponentCodes | array of Integer | Массив кодов комплектующих в конфигурации | TPCBuildOption |
| TotalPrice | Real | Общая стоимость конфигурации | TPCBuildOption |
| ID | Integer | Уникальный идентификатор заказа | TOrder |
| BuildOptionID | Integer | ID выбранного варианта комплектации | TOrder |
| CustomerName | TFixedString | Имя заказчика | TOrder |
| CustomerPhone | TFixedString | Телефон заказчика | TOrder |
| OrderDate | TDateTime | Дата оформления заказа | TOrder |
| Data | TComponentType | Данные о типе комплектующей | TComponentTypeNode |
| Next | PComponentTypeNode | Указатель на следующий узел | TComponentTypeNode |
| Data | TComponent | Данные о комплектующей | TComponentNode |
| Next | PComponentNode | Указатель на следующий узел | TComponentNode |
| Data | TCompatibility | Данные о совместимости | TCompatibilityNode |
| Next | PCompatibilityNode | Указатель на следующий узел | TCompatibilityNode |
| Data | TPCBuildOption | Данные о варианте комплектации | TPCBuildOptionNode |
| Next | PPCBuildOptionNode | Указатель на следующий узел | TPCBuildOptionNode |
| Data | TOrder | Данные о заказе | TOrderNode |
| Next | POrderNode | Указатель на следующий узел | TOrderNode |
| Head | PComponentTypeNode | Указатель на первый элемент списка | TComponentTypeList |
| Count | Integer | Количество элементов в списке | TComponentTypeList |
| Head | PComponentNode | Указатель на первый элемент списка | TComponentList |
| Count | Integer | Количество элементов в списке | TComponentList |
| Head | PCompatibilityNode | Указатель на первый элемент списка | TCompatibilityList |
| Count | Integer | Количество элементов в списке | TCompatibilityList |
| Head | PPCBuildOptionNode | Указатель на первый элемент списка | TPCBuildOptionList |
| Count | Integer | Количество элементов в списке | TPCBuildOptionList |
| Head | POrderNode | Указатель на первый элемент списка | TOrderList |
| Count | Integer | Количество элементов в списке | TOrderList |

### 

### 3.4. Определение подпрограмм и их описание

Таблица 3.3 Используемые идентификаторы

| Таблица 3.3 – Продолжение | | | |
| --- | --- | --- | --- |
| Имя подпрограммы | Назначение подпрограммы | Заголовок подпрограммы | Имя параметра, его тип, назначение |
| InitAllLists | Инициализирует все списки данных программы | procedure InitAllLists(var ComponentTypesList, ComponentsList, CompatibilityList, PCBuildOptionsList, OrdersList) | ComponentTypesList: TComponentTypeList — список типов комплектующих; ComponentsList: TComponentList — список комплектующих; CompatibilityList: TCompatibilityList — список совместимости; PCBuildOptionsList: TPCBuildOptionList — список вариантов сборки; OrdersList: TOrderList — список заказов |
| LoadAllData | Загружает все данные из файлов с запросом подтверждения у пользователя | procedure LoadAllData(var ComponentTypesList, ComponentsList, CompatibilityList, OrdersList) | ComponentTypesList: TComponentTypeList — список типов комплектующих; ComponentsList: TComponentList — список комплектующих; CompatibilityList: TCompatibilityList — список совместимости; OrdersList: TOrderList — список заказов |
| SaveAllData | Сохраняет все данные в соответствующие файлы | procedure SaveAllData(const ComponentTypesList, ComponentsList, CompatibilityList, OrdersList) | ComponentTypesList: TComponentTypeList — список типов комплектующих; ComponentsList: TComponentList — список комплектующих; CompatibilityList: TCompatibilityList — список совместимости; OrdersList: TOrderList — список заказов |
| ViewList | Отображает выбранный список данных через подменю | procedure ViewList(const ComponentTypesList, ComponentsList, CompatibilityList, OrdersList, PCBuildOptionsList) | ComponentTypesList: TComponentTypeList — список типов комплектующих; ComponentsList: TComponentList — список комплектующих; CompatibilityList: TCompatibilityList — список совместимости; OrdersList: TOrderList — список заказов; PCBuildOptionsList: TPCBuildOptionList — список вариантов сборки |
| AddData | Добавляет новые данные в выбранный список через подменю | procedure AddData(var ComponentTypesList, ComponentsList, CompatibilityList) | ComponentTypesList: TComponentTypeList — список типов комплектующих; ComponentsList: TComponentList — список комплектующих; CompatibilityList: TCompatibilityList — список совместимости |
| RemoveData | Удаляет данные из выбранного списка с проверкой зависимостей | procedure RemoveData(var ComponentTypesList, ComponentsList, CompatibilityList) | ComponentTypesList: TComponentTypeList — список типов комплектующих; ComponentsList: TComponentList — список комплектующих; CompatibilityList: TCompatibilityList — список совместимости |
| EditData | Редактирует данные в выбранном списке | procedure EditData(var ComponentTypesList, ComponentsList, CompatibilityList) | ComponentTypesList: TComponentTypeList — список типов комплектующих; ComponentsList: TComponentList — список комплектующих; CompatibilityList: TCompatibilityList — список совместимости |
| HandleSpecialFunctions | Обрабатывает специальные функции: подбор ПК, заказы, поиск совместимости | procedure HandleSpecialFunctions(const ComponentTypesList, ComponentsList, CompatibilityList, var PCBuildOptionsList, OrdersList) | ComponentTypesList: TComponentTypeList — список типов комплектующих; ComponentsList: TComponentList — список комплектующих; CompatibilityList: TCompatibilityList — список совместимости; PCBuildOptionsList: TPCBuildOptionList — список вариантов сборки; OrdersList: TOrderList — список заказов |
| HandleSortData | Сортирует данные выбранного типа по заданному критерию | procedure HandleSortData(var ComponentTypesList, ComponentsList, OrdersList) | ComponentTypesList: TComponentTypeList — список типов комплектующих; ComponentsList: TComponentList — список комплектующих; OrdersList: TOrderList — список заказов |
| HandleSearchData | Выполняет поиск комплектующих по различным фильтрам | procedure HandleSearchData(const ComponentsList, ComponentTypesList) | ComponentsList: TComponentList — список комплектующих; ComponentTypesList: TComponentTypeList — список типов комплектующих |
| FindPCBuildOptions | Находит все возможные варианты сборки ПК в заданном ценовом диапазоне | function FindPCBuildOptions(const ComponentList, CompatibilityList, TypesList; MinPrice, MaxPrice: Real; var BuildOptionsList): Boolean | ComponentList: TComponentList — список комплектующих; CompatibilityList: TCompatibilityList — список совместимости; TypesList: TComponentTypeList — список типов; MinPrice, MaxPrice: Real — ценовой диапазон; BuildOptionsList: TPCBuildOptionList — результирующий список вариантов |
| CreateOrder | Оформляет заказ на основе выбранного варианта комплектации | function CreateOrder(const BuildOption; const ComponentList; var OrdersList): Boolean | BuildOption: TPCBuildOption — выбранный вариант сборки; ComponentList: TComponentList — список комплектующих; OrdersList: TOrderList — список заказов |
| FindCompatibleComponents | Находит комплектующие заданного типа, совместимые с указанной комплектующей | function FindCompatibleComponents(const ComponentList, CompatibilityList, TypesList; ComponentCode, TypeCode: Integer; var CompatibleComponents): Boolean | ComponentList: TComponentList — список комплектующих; CompatibilityList: TCompatibilityList — список совместимости; TypesList: TComponentTypeList — список типов; ComponentCode, TypeCode: Integer — коды базовой комплектующей и искомого типа; CompatibleComponents: TComponentList — результирующий список |
| DisplayMainMenu | Отображает главное меню программы | procedure DisplayMainMenu | Без параметров |
| DisplayComponents | Отображает список комплектующих с информацией о типах | procedure DisplayComponents(const List; const TypesList) | List: TComponentList — список комплектующих; TypesList: TComponentTypeList — список типов для отображения названий |
| InputComponent | Выполняет ввод данных о новой комплектующей с проверкой | function InputComponent(var Component; const TypesList, List): Boolean | Component: TComponent — структура для заполнения; TypesList: TComponentTypeList — список типов; List: TComponentList — список комплектующих для проверки уникальности |
| SaveOrderToTextFile | Сохраняет детальную информацию о заказе в текстовый файл | function SaveOrderToTextFile(const Order, BuildOption; const ComponentList; const FileName): Boolean | Order: TOrder — данные заказа; BuildOption: TPCBuildOption — данные варианта сборки; ComponentList: TComponentList — список комплектующих; FileName: string — имя файла |
| SafeReadInteger | Безопасный ввод целого числа с проверкой диапазона и команд отмены | function SafeReadInteger(const Prompt: string; MinValue, MaxValue: Integer; const ErrorMsg: string): Integer | Prompt: string — приглашение к вводу; MinValue, MaxValue: Integer — диапазон допустимых значений; ErrorMsg: string — сообщение об ошибке |

### 3.5 Тестирование и отладка программы

Таблица 3.4 Прохождение тестов программы

| Таблица 3.4 – Продолжение | | | |
| --- | --- | --- | --- |
| Номер теста | Вводимые данные | Ожидаемый результат | Полученный результат |
| 1 | Выбор основных действий главного меню программы (пункты 1-10) | Корректное открытие соответствующих функций и подменю для каждого пункта | Тест пройден |
| 2 | Загрузка данных из файлов (пункт меню 1) при отсутствии файлов данных | Сообщения о создании пустых списков для каждого типа данных, успешная инициализация программы | Тест пройден |
| 3 | Ввод корректных данных при добавлении типа комплектующих (код: 1, название: “Процессор”) | Автоматическая генерация ID, успешное добавление записи, сообщение о добавлении | Тест пройден |
| 4 | Ввод некорректных данных при добавлении комплектующих (отрицательная цена, пустое название производителя) | Вывод сообщений об ошибках валидации, повторный запрос корректных данных | Тест пройден |
| 5 | Попытка добавить комплектующую с уже существующим кодом | Сообщение о существовании комплектующей с таким кодом, требование ввести другой код | Тест пройден |
| 6 | Добавление правила совместимости между несуществующими комплектующими | Сообщение об ошибке, указание на отсутствие комплектующих с указанными кодами | Тест пройден |
| 7 | Поиск комплектующих по диапазону цен (минимум: 1000, максимум: 5000) | Отображение всех комплектующих, цена которых находится в указанном диапазоне | Тест пройден |
| 8 | Ввод некорректного ценового диапазона (максимум меньше минимума) | Сообщение об ошибке, повторный запрос корректного диапазона | Тест пройден |
| 9 | Сортировка списка комплектующих по цене | Отображение списка комплектующих в порядке возрастания цены | Тест пройден |
| 10 | Попытка удаления типа комплектующих, для которого существуют зависимые комплектующие | Сообщение о невозможности удаления из-за существующих зависимостей | Тест пройден |
| 11 | Редактирование цены существующей комплектующей на корректное значение | Успешное обновление цены, сообщение о внесенных изменениях | Тест пройден |
| 12 | Подбор вариантов сборки ПК в ценовом диапазоне при наличии совместимых комплектующих | Отображение найденных вариантов сборки, отсортированных по цене, сохранение в файл | Тест пройден |
| 13 | Подбор вариантов сборки ПК при отсутствии совместимых комплектующих в диапазоне | Сообщение о том, что варианты не найдены | Тест пройден |
| 14 | Оформление заказа на существующий вариант сборки с достаточным количеством на складе | Успешное создание заказа, уменьшение количества товара на складе, создание файла заказа | Тест пройден |
| 15 | Попытка оформления заказа при недостаточном количестве товара на складе | Сообщение об ошибке недостатка товара, отказ в создании заказа | Тест пройден |
| 16 | Поиск совместимых комплектующих заданного типа для существующей комплектующей | Отображение списка совместимых комплектующих указанного типа | Тест пройден |
| 17 | Использование команд отмены (“/cancel”, “/menu”) при вводе данных | Корректный возврат в предыдущее меню без сохранения введенных данных | Тест пройден |
| 18 | Сохранение всех изменений при выходе (пункт меню 10) | Успешная запись всех списков в соответствующие файлы, подтверждение сохранения | Тест пройден |
| 19 | Выход без сохранения (пункт меню 9) | Завершение программы без записи изменений в файлы, корректное освобождение памяти | Тест пройден |
| 20 | Просмотр всех списков данных через подменю (типы, комплектующие, совместимость, заказы) | Корректное отображение содержимого каждого списка с подходящим форматированием | Тест пройден |
| 21 | Ввод слишком длинных строк при добавлении данных (превышение MAX\_STRING\_LENGTH) | Обрезание строк до максимальной длины или сообщение об ошибке | Тест пройден |
| 22 | Работа с пустыми списками (попытка сортировки, поиска, удаления) | Корректные сообщения о том, что операция невозможна на пустых списках | Тест пройден |
| 23 | Поиск несуществующих элементов по различным критериям | Сообщения о том, что элементы не найдены, без системных ошибок | Тест пройден |
| 24 | Ввод специальных символов и некорректных данных в различных полях | Корректная валидация и отклонение некорректных данных с понятными сообщениями | Тест пройден |
| 25 | Последовательное выполнение полного цикла операций (загрузка → добавление → редактирование → поиск → сохранение) | Корректная работа всех компонентов программы в связке, сохранение целостности данных | Тест пройден |

# 4. РУКОВОДСТВО ПО УСТАНОВКЕ И ЭКСПЛУАТАЦИИ

### 4.1. Установка программы

1. Системные требования:
   * Операционная система: Linux, macOS, Windows
   * Консольное приложение с минимальными требованиями к ресурсам системы
2. Процесс установки:
   * Для операционной системы macOS достаточно
     + - * Выдать исполняемому файлу права в файловой системе (chmod 777 bin\_solve)
         * Запустить исполняемый файл (./bin\_solve)
   * При необходимости сборки:
     + - * Установить компилятор fpc
         * Установить утилиту make (обычно предустановлена)
         * Запустить команду make build из корня проекта
         * Выдать исполняемому файлу права в файловой системе (chmod 777 bin\_solve)
         * Запустить исполняемый файл (./bin\_solve)
   * Дополнительные файлы перед первым запуском не требуются
   * При первом запуске и последующем сохранении данных программа автоматически создаст необходимые типизированные файлы (components.dat, component\_types.dat, compatibility.dat, orders.dat) в директории data/
   * При выполнении специальных функций будут генерироваться текстовые файлы с результатами подбора ПК и информацией о заказах

### 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 всего списка
3. Сортировка данных
4. Поиск данных по различным критериям
5. Добавление данных в список
6. Удаление данных из списка
7. Редактирование данных
8. Сборка ПК и Заказы (специальные функции)
9. Выход c cохранением
10. Выход

### 4.2.2. Основные функции программы

Загрузка данных:

• Загрузка данных запрашивает подтверждение пользователя перед заменой текущих данных в памяти

• Если файлы данных отсутствуют, программа начнет работу с пустыми списками

• Загружаются типы комплектующих, комплектующие, правила совместимости и заказы

Просмотр данных:

• Пункт 2 позволяет просматривать списки типов комплектующих, комплектующих, правил совместимости и заказов

• Реализован постраничный вывод данных при большом объеме информации

Управление типами комплектующих:

• Добавление типов (пункт 5): ввод кода типа и названия (например, процессор, видеокарта)

• Редактирование типов (пункт 7): изменение названия типа комплектующих

• Удаление типов (пункт 6): удаление возможно только если тип не используется в комплектующих

Управление комплектующими:

• Добавление комплектующих (пункт 5): ввод кода, типа, производителя, модели, параметров, цены и количества на складе

• Редактирование комплектующих (пункт 7): изменение любых параметров существующей комплектующей

• Удаление комплектующих (пункт 6): удаление возможно только если комплектующая не используется в правилах совместимости

Управление совместимостью:

• Добавление правил совместимости (пункт 5): указание двух совместимых комплектующих

• Редактирование правил (пункт 7): изменение связей между комплектующими

• Удаление правил (пункт 6): удаление существующих правил совместимости

Специальные функции (пункт 8):

• Подбор конфигурации ПК в заданном диапазоне цен: автоматический поиск всех возможных сборок из совместимых комплектующих

• Поиск совместимых комплектующих: поиск компонентов заданного типа, совместимых с выбранным компонентом

• Оформление заказа: выбор понравившейся конфигурации и создание заказа с генерацией информационного файла

Поиск и сортировка:

• Поиск данных (пункт 4): поиск по различным критериям в зависимости от типа данных

• Сортировка данных (пункт 3): сортировка списков по различным параметрам (цена, код, производитель)

Сохранение данных:

• Для сохранения всех изменений используйте пункт 9

• Данные сохраняются в типизированные файлы components.dat, component\_types.dat, compatibility.dat и orders.dat в директории data/

### 4.2.3 Форматы ввода данных

Дата: формат ДД.ММ.ГГГГ

• Дата должна быть действительной и корректной

Коды элементов: положительные целые числа

• Коды типов комплектующих, комплектующих должны быть уникальными в пределах каждого типа

Текстовые поля: ограничения по длине

• Производитель: до 30 символов

• Модель комплектующих: до 50 символов

• Параметры и характеристики: до 100 символов

• Название типа: до 30 символов

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

• Цена должна быть больше нуля

Количество на складе: неотрицательное целое число

### 4.2.4 Работа с файлами

• Программа автоматически создает и обновляет типизированные файлы данных при сохранении (пункт 9 главного меню)

• При подборе конфигураций ПК генерируются текстовые файлы с результатами (PC\_Config\_[номер].txt)

• При оформлении заказа создается текстовый файл с детальной информацией о заказе (Order\_[номер].txt)

• Файлы данных (components.dat, component\_types.dat, compatibility.dat, orders.dat) имеют бинарный формат и не предназначены для ручного редактирования

• Все файлы данных сохраняются в директории data/ относительно исполняемого файла программы

### 4.3 Возможные проблемы и их решения

• Проблема: Программа сообщает, что невозможно загрузить данные. Решение: Убедитесь, что файлы данных находятся в директории data/ относительно исполняемого файла программы.

• Проблема: Невозможно удалить тип комплектующих. Решение: Тип не может быть удален, если для него существуют комплектующие. Сначала удалите все связанные комплектующие.

• Проблема: Невозможно удалить комплектующую. Решение: Комплектующая не может быть удалена, если она используется в правилах совместимости. Сначала удалите все связанные правила совместимости.

• Проблема: Программа не находит данные при поиске. Решение: Убедитесь, что вы вводите критерии поиска точно так же, как они были введены при создании записей (включая регистр букв).

• Проблема: Невозможно подобрать конфигурацию ПК. Решение: Убедитесь, что в системе есть комплектующие разных типов и между ними установлены правила совместимости.

• Проблема: Не удается оформить заказ. Решение: Проверьте, что выбранная конфигурация существует в списке подобранных вариантов и все комплектующие доступны на складе.

• Проблема: Ошибка при сохранении данных. Решение: Убедитесь, что у программы есть права на запись в директорию data/ и достаточно свободного места на диске.

# Приложение 1. Код программы

======== DataSearch.pas ========

unit DataSearch;

{$mode objfpc}{$H+}

interface

uses

SysUtils,

DataTypes,

DynamicLists;

// Функции поиска

procedure SearchComponentsByPriceRange(const SourceList: TComponentList; MinPrice, MaxPrice: Real; var ResultList: TComponentList);

procedure SearchComponentsByManufacturer(const SourceList: TComponentList; const Manufacturer: string; var ResultList: TComponentList);

procedure SearchComponentsByType(const SourceList: TComponentList; TypeCode: Integer; var ResultList: TComponentList);

procedure SearchComponentsByModel(const SourceList: TComponentList; const Model: string; var ResultList: TComponentList);

procedure SearchComponentsByMinStock(const SourceList: TComponentList; MinStock: Integer; var ResultList: TComponentList);

implementation

// Функции поиска

procedure SearchComponentsByPriceRange(const SourceList: TComponentList; MinPrice, MaxPrice: Real; var ResultList: TComponentList);

var

Current: PComponentNode;

begin

Current := SourceList.Head;

while Current <> nil do

begin

if (Current^.Data.Price >= MinPrice) and (Current^.Data.Price <= MaxPrice) then

AddComponent(ResultList, Current^.Data);

Current := Current^.Next;

end;

end;

procedure SearchComponentsByManufacturer(const SourceList: TComponentList; const Manufacturer: string; var ResultList: TComponentList);

var

Current: PComponentNode;

CurrentManufacturer: string;

begin

Current := SourceList.Head;

while Current <> nil do

begin

CurrentManufacturer := FixedToString(Current^.Data.Manufacturer);

if Pos(LowerCase(Manufacturer), LowerCase(CurrentManufacturer)) > 0 then

AddComponent(ResultList, Current^.Data);

Current := Current^.Next;

end;

end;

procedure SearchComponentsByType(const SourceList: TComponentList; TypeCode: Integer; var ResultList: TComponentList);

var

Current: PComponentNode;

begin

Current := SourceList.Head;

while Current <> nil do

begin

if Current^.Data.TypeCode = TypeCode then

AddComponent(ResultList, Current^.Data);

Current := Current^.Next;

end;

end;

procedure SearchComponentsByModel(const SourceList: TComponentList; const Model: string; var ResultList: TComponentList);

var

Current: PComponentNode;

CurrentModel: string;

begin

Current := SourceList.Head;

while Current <> nil do

begin

CurrentModel := FixedToString(Current^.Data.Model);

if Pos(LowerCase(Model), LowerCase(CurrentModel)) > 0 then

AddComponent(ResultList, Current^.Data);

Current := Current^.Next;

end;

end;

procedure SearchComponentsByMinStock(const SourceList: TComponentList; MinStock: Integer; var ResultList: TComponentList);

var

Current: PComponentNode;

begin

Current := SourceList.Head;

while Current <> nil do

begin

if Current^.Data.InStock >= MinStock then

AddComponent(ResultList, Current^.Data);

Current := Current^.Next;

end;

end;

end.

======== DataSorting.pas ========

unit DataSorting;

{$mode objfpc}{$H+}

interface

uses

DataTypes,

DynamicLists;

// Функции сортировки

procedure SortComponentTypesByCode(var List: TComponentTypeList);

procedure SortComponentsByPrice(var List: TComponentList);

procedure SortComponentsByCode(var List: TComponentList);

procedure SortComponentsByManufacturer(var List: TComponentList);

procedure SortOrdersByDate(var List: TOrderList);

implementation

procedure SortComponentTypesByCode(var List: TComponentTypeList);

var

i, j: Integer;

CurrentI: PComponentTypeNode;

TempData: TComponentType;

begin

if (List.Head = nil) or (List.Head^.Next = nil) then

Exit; // Список пуст или содержит только один элемент

// Пузырьковая сортировка для односвязного списка

for i := 0 to List.Count - 2 do

begin

CurrentI := List.Head;

for j := 0 to List.Count - 2 - i do

begin

if (CurrentI^.Next <> nil) and (CurrentI^.Data.TypeCode > CurrentI^.Next^.Data.TypeCode) then

begin

// Обмен данными между узлами

TempData := CurrentI^.Data;

CurrentI^.Data := CurrentI^.Next^.Data;

CurrentI^.Next^.Data := TempData;

end;

CurrentI := CurrentI^.Next;

if CurrentI = nil then

Break;

end;

end;

end;

procedure SortComponentsByPrice(var List: TComponentList);

var

i, j: Integer;

CurrentI: PComponentNode;

TempData: TComponent;

begin

if (List.Head = nil) or (List.Head^.Next = nil) then

Exit;

for i := 0 to List.Count - 2 do

begin

CurrentI := List.Head;

for j := 0 to List.Count - 2 - i do

begin

if (CurrentI^.Next <> nil) and (CurrentI^.Data.Price > CurrentI^.Next^.Data.Price) then

begin

TempData := CurrentI^.Data;

CurrentI^.Data := CurrentI^.Next^.Data;

CurrentI^.Next^.Data := TempData;

end;

CurrentI := CurrentI^.Next;

if CurrentI = nil then

Break;

end;

end;

end;

procedure SortComponentsByCode(var List: TComponentList);

var

i, j: Integer;

CurrentI: PComponentNode;

TempData: TComponent;

begin

if (List.Head = nil) or (List.Head^.Next = nil) then

Exit;

for i := 0 to List.Count - 2 do

begin

CurrentI := List.Head;

for j := 0 to List.Count - 2 - i do

begin

if (CurrentI^.Next <> nil) and (CurrentI^.Data.Code > CurrentI^.Next^.Data.Code) then

begin

TempData := CurrentI^.Data;

CurrentI^.Data := CurrentI^.Next^.Data;

CurrentI^.Next^.Data := TempData;

end;

CurrentI := CurrentI^.Next;

if CurrentI = nil then

Break;

end;

end;

end;

procedure SortComponentsByManufacturer(var List: TComponentList);

var

i, j: Integer;

CurrentI: PComponentNode;

TempData: TComponent;

begin

if (List.Head = nil) or (List.Head^.Next = nil) then

Exit;

for i := 0 to List.Count - 2 do

begin

CurrentI := List.Head;

for j := 0 to List.Count - 2 - i do

begin

if (CurrentI^.Next <> nil) and

(FixedToString(CurrentI^.Data.Manufacturer) > FixedToString(CurrentI^.Next^.Data.Manufacturer)) then

begin

TempData := CurrentI^.Data;

CurrentI^.Data := CurrentI^.Next^.Data;

CurrentI^.Next^.Data := TempData;

end;

CurrentI := CurrentI^.Next;

if CurrentI = nil then

Break;

end;

end;

end;

procedure SortOrdersByDate(var List: TOrderList);

var

i, j: Integer;

CurrentI: POrderNode;

TempData: TOrder;

begin

if (List.Head = nil) or (List.Head^.Next = nil) then

Exit;

for i := 0 to List.Count - 2 do

begin

CurrentI := List.Head;

for j := 0 to List.Count - 2 - i do

begin

if (CurrentI^.Next <> nil) and (CurrentI^.Data.OrderDate > CurrentI^.Next^.Data.OrderDate) then

begin

TempData := CurrentI^.Data;

CurrentI^.Data := CurrentI^.Next^.Data;

CurrentI^.Next^.Data := TempData;

end;

CurrentI := CurrentI^.Next;

if CurrentI = nil then

Break;

end;

end;

end;

end.

======== DataTypes.pas ========

unit DataTypes;

{$mode objfpc}{$H+}

interface

uses

SysUtils;

const

MAX\_STRING\_LENGTH = 255;

type

{ Строка фиксированной длины для использования в типизированных файлах }

TFixedString = array[0..MAX\_STRING\_LENGTH] of Char;

{ Предварительные объявления типов указателей }

PComponentType = ^TComponentType;

PComponent = ^TComponent;

PCompatibility = ^TCompatibility;

PPCBuildOption = ^TPCBuildOption;

POrder = ^TOrder;

PComponentTypeNode = ^TComponentTypeNode;

PComponentNode = ^TComponentNode;

PCompatibilityNode = ^TCompatibilityNode;

PPCBuildOptionNode = ^TPCBuildOptionNode;

POrderNode = ^TOrderNode;

{ Тип комплектующей }

TComponentType = record

TypeCode: Integer; // Код типа комплектующей

Name: TFixedString; // Название типа (например, "Процессор", "Видеокарта" и т.д.)

end;

{ Комплектующая }

TComponent = record

Code: Integer; // Уникальный код комплектующей

TypeCode: Integer; // Код типа комплектующей (связь с TComponentType)

Manufacturer: TFixedString; // Фирма-изготовитель

Model: TFixedString; // Модель

Parameters: TFixedString; // Параметры (строка с описанием характеристик)

Price: Real; // Цена

InStock: Integer; // Количество в наличии

end;

{ Совместимость комплектующих }

TCompatibility = record

ComponentCode1: Integer; // Код первой комплектующей

ComponentCode2: Integer; // Код второй комплектующей (совместимой с первой)

end;

{ Вариант комплектации ПК }

TPCBuildOption = record

ID: Integer; // Уникальный идентификатор варианта

ComponentCodes: array of Integer; // Массив кодов комплектующих в этой конфигурации

TotalPrice: Real; // Общая стоимость конфигурации

end;

{ Заказ }

TOrder = record

ID: Integer; // Уникальный идентификатор заказа

BuildOptionID: Integer; // ID выбранного варианта комплектации

CustomerName: TFixedString; // Имя заказчика

CustomerPhone: TFixedString; // Телефон заказчика

OrderDate: TDateTime; // Дата заказа

end;

{ Узел односвязного списка для TComponentType }

TComponentTypeNode = record

Data: TComponentType;

Next: PComponentTypeNode;

end;

{ Узел односвязного списка для TComponent }

TComponentNode = record

Data: TComponent;

Next: PComponentNode;

end;

{ Узел односвязного списка для TCompatibility }

TCompatibilityNode = record

Data: TCompatibility;

Next: PCompatibilityNode;

end;

{ Узел односвязного списка для TPCBuildOption }

TPCBuildOptionNode = record

Data: TPCBuildOption;

Next: PPCBuildOptionNode;

end;

{ Узел односвязного списка для TOrder }

TOrderNode = record

Data: TOrder;

Next: POrderNode;

end;

{ Списки для каждого типа данных }

TComponentTypeList = record

Head: PComponentTypeNode;

Count: Integer;

end;

TComponentList = record

Head: PComponentNode;

Count: Integer;

end;

TCompatibilityList = record

Head: PCompatibilityNode;

Count: Integer;

end;

TPCBuildOptionList = record

Head: PPCBuildOptionNode;

Count: Integer;

end;

TOrderList = record

Head: POrderNode;

Count: Integer;

end;

{ Функции для работы с фиксированными строками }

function StringToFixed(const S: string): TFixedString;

function FixedToString(const F: TFixedString): string;

implementation

{ Функции для работы с фиксированными строками }

function StringToFixed(const S: string): TFixedString;

var

i: Integer;

begin

FillChar(Result, SizeOf(Result), 0);

for i := 1 to Length(S) do

begin

if i > MAX\_STRING\_LENGTH then

Break;

Result[i-1] := S[i];

end;

end;

function FixedToString(const F: TFixedString): string;

var

i: Integer;

begin

Result := '';

i := 0;

while (i <= MAX\_STRING\_LENGTH) and (F[i] <> #0) do

begin

Result := Result + F[i];

Inc(i);

end;

end;

end.

======== DynamicLists.pas ========

unit DynamicLists;

{$mode objfpc}{$H+}

interface

uses

SysUtils, DataTypes;

{ Функции для работы со списком типов комплектующих }

procedure InitComponentTypeList(var List: TComponentTypeList);

function AddComponentType(var List: TComponentTypeList; const Data: TComponentType): Boolean;

function RemoveComponentType(var List: TComponentTypeList; TypeCode: Integer): Boolean;

function FindComponentType(const List: TComponentTypeList; TypeCode: Integer): PComponentTypeNode;

procedure ClearComponentTypeList(var List: TComponentTypeList);

{ Функции для работы со списком комплектующих }

procedure InitComponentList(var List: TComponentList);

function AddComponent(var List: TComponentList; const Data: TComponent): Boolean;

function RemoveComponent(var List: TComponentList; Code: Integer): Boolean;

function FindComponent(const List: TComponentList; Code: Integer): PComponentNode;

procedure ClearComponentList(var List: TComponentList);

{ Функции для работы со списком совместимости }

procedure InitCompatibilityList(var List: TCompatibilityList);

function AddCompatibility(var List: TCompatibilityList; const Data: TCompatibility): Boolean;

function RemoveCompatibility(var List: TCompatibilityList; ComponentCode1, ComponentCode2: Integer): Boolean;

function FindCompatibility(const List: TCompatibilityList; ComponentCode1, ComponentCode2: Integer): PCompatibilityNode;

procedure ClearCompatibilityList(var List: TCompatibilityList);

{ Функции для работы со списком вариантов комплектации }

procedure InitPCBuildOptionList(var List: TPCBuildOptionList);

function AddPCBuildOption(var List: TPCBuildOptionList; const Data: TPCBuildOption): Boolean;

function RemovePCBuildOption(var List: TPCBuildOptionList; ID: Integer): Boolean;

function FindPCBuildOption(const List: TPCBuildOptionList; ID: Integer): PPCBuildOptionNode;

procedure ClearPCBuildOptionList(var List: TPCBuildOptionList);

{ Функции для работы со списком заказов }

procedure InitOrderList(var List: TOrderList);

function AddOrder(var List: TOrderList; const Data: TOrder): Boolean;

function RemoveOrder(var List: TOrderList; ID: Integer): Boolean;

function FindOrder(const List: TOrderList; ID: Integer): POrderNode;

procedure ClearOrderList(var List: TOrderList);

implementation

{ Реализация функций для работы со списком типов комплектующих }

procedure InitComponentTypeList(var List: TComponentTypeList);

begin

List.Head := nil;

List.Count := 0;

end;

function AddComponentType(var List: TComponentTypeList; const Data: TComponentType): Boolean;

var

NewNode: PComponentTypeNode;

begin

Result := False;

// Проверяем, не существует ли уже тип с таким кодом

if FindComponentType(List, Data.TypeCode) <> nil then

Exit;

// Создаем новый узел

New(NewNode);

NewNode^.Data := Data;

NewNode^.Next := List.Head;

// Добавляем в начало списка

List.Head := NewNode;

Inc(List.Count);

Result := True;

end;

function RemoveComponentType(var List: TComponentTypeList; TypeCode: Integer): Boolean;

var

Current, Previous: PComponentTypeNode;

begin

Result := False;

if List.Head = nil then

Exit;

// Если удаляемый элемент - первый в списке

if List.Head^.Data.TypeCode = TypeCode then

begin

Current := List.Head;

List.Head := List.Head^.Next;

Dispose(Current);

Dec(List.Count);

Result := True;

Exit;

end;

// Ищем элемент в списке

Previous := List.Head;

Current := List.Head^.Next;

while (Current <> nil) and (Current^.Data.TypeCode <> TypeCode) do

begin

Previous := Current;

Current := Current^.Next;

end;

// Если элемент найден

if Current <> nil then

begin

Previous^.Next := Current^.Next;

Dispose(Current);

Dec(List.Count);

Result := True;

end;

end;

function FindComponentType(const List: TComponentTypeList; TypeCode: Integer): PComponentTypeNode;

var

Current: PComponentTypeNode;

begin

Result := nil;

Current := List.Head;

while Current <> nil do

begin

if Current^.Data.TypeCode = TypeCode then

begin

Result := Current;

Exit;

end;

Current := Current^.Next;

end;

end;

procedure ClearComponentTypeList(var List: TComponentTypeList);

var

Current, Next: PComponentTypeNode;

begin

Current := List.Head;

while Current <> nil do

begin

Next := Current^.Next;

Dispose(Current);

Current := Next;

end;

List.Head := nil;

List.Count := 0;

end;

{ Реализация функций для работы со списком комплектующих }

procedure InitComponentList(var List: TComponentList);

begin

List.Head := nil;

List.Count := 0;

end;

function AddComponent(var List: TComponentList; const Data: TComponent): Boolean;

var

NewNode: PComponentNode;

begin

Result := False;

// Проверяем, не существует ли уже комплектующая с таким кодом

if FindComponent(List, Data.Code) <> nil then

Exit;

// Создаем новый узел

New(NewNode);

NewNode^.Data := Data;

NewNode^.Next := List.Head;

// Добавляем в начало списка

List.Head := NewNode;

Inc(List.Count);

Result := True;

end;

function RemoveComponent(var List: TComponentList; Code: Integer): Boolean;

var

Current, Previous: PComponentNode;

begin

Result := False;

if List.Head = nil then

Exit;

// Если удаляемый элемент - первый в списке

if List.Head^.Data.Code = Code then

begin

Current := List.Head;

List.Head := List.Head^.Next;

Dispose(Current);

Dec(List.Count);

Result := True;

Exit;

end;

// Ищем элемент в списке

Previous := List.Head;

Current := List.Head^.Next;

while (Current <> nil) and (Current^.Data.Code <> Code) do

begin

Previous := Current;

Current := Current^.Next;

end;

// Если элемент найден

if Current <> nil then

begin

Previous^.Next := Current^.Next;

Dispose(Current);

Dec(List.Count);

Result := True;

end;

end;

function FindComponent(const List: TComponentList; Code: Integer): PComponentNode;

var

Current: PComponentNode;

begin

Result := nil;

Current := List.Head;

while Current <> nil do

begin

if Current^.Data.Code = Code then

begin

Result := Current;

Exit;

end;

Current := Current^.Next;

end;

end;

procedure ClearComponentList(var List: TComponentList);

var

Current, Next: PComponentNode;

begin

Current := List.Head;

while Current <> nil do

begin

Next := Current^.Next;

Dispose(Current);

Current := Next;

end;

List.Head := nil;

List.Count := 0;

end;

{ Реализация функций для работы со списком совместимости }

procedure InitCompatibilityList(var List: TCompatibilityList);

begin

List.Head := nil;

List.Count := 0;

end;

function AddCompatibility(var List: TCompatibilityList; const Data: TCompatibility): Boolean;

var

NewNode: PCompatibilityNode;

begin

Result := False;

// Проверяем, не существует ли уже такая запись о совместимости

if FindCompatibility(List, Data.ComponentCode1, Data.ComponentCode2) <> nil then

Exit;

// Создаем новый узел

New(NewNode);

NewNode^.Data := Data;

NewNode^.Next := List.Head;

// Добавляем в начало списка

List.Head := NewNode;

Inc(List.Count);

Result := True;

end;

function RemoveCompatibility(var List: TCompatibilityList; ComponentCode1, ComponentCode2: Integer): Boolean;

var

Current, Previous: PCompatibilityNode;

begin

Result := False;

if List.Head = nil then

Exit;

// Если удаляемый элемент - первый в списке

if (List.Head^.Data.ComponentCode1 = ComponentCode1) and

(List.Head^.Data.ComponentCode2 = ComponentCode2) then

begin

Current := List.Head;

List.Head := List.Head^.Next;

Dispose(Current);

Dec(List.Count);

Result := True;

Exit;

end;

// Ищем элемент в списке

Previous := List.Head;

Current := List.Head^.Next;

while (Current <> nil) and

((Current^.Data.ComponentCode1 <> ComponentCode1) or

(Current^.Data.ComponentCode2 <> ComponentCode2)) do

begin

Previous := Current;

Current := Current^.Next;

end;

// Если элемент найден

if Current <> nil then

begin

Previous^.Next := Current^.Next;

Dispose(Current);

Dec(List.Count);

Result := True;

end;

end;

function FindCompatibility(const List: TCompatibilityList; ComponentCode1, ComponentCode2: Integer): PCompatibilityNode;

var

Current: PCompatibilityNode;

begin

Result := nil;

Current := List.Head;

while Current <> nil do

begin

if (Current^.Data.ComponentCode1 = ComponentCode1) and

(Current^.Data.ComponentCode2 = ComponentCode2) then

begin

Result := Current;

Exit;

end;

Current := Current^.Next;

end;

end;

procedure ClearCompatibilityList(var List: TCompatibilityList);

var

Current, Next: PCompatibilityNode;

begin

Current := List.Head;

while Current <> nil do

begin

Next := Current^.Next;

Dispose(Current);

Current := Next;

end;

List.Head := nil;

List.Count := 0;

end;

{ Реализация функций для работы со списком вариантов комплектации }

procedure InitPCBuildOptionList(var List: TPCBuildOptionList);

begin

List.Head := nil;

List.Count := 0;

end;

function AddPCBuildOption(var List: TPCBuildOptionList; const Data: TPCBuildOption): Boolean;

var

NewNode: PPCBuildOptionNode;

begin

Result := False;

// Проверяем, не существует ли уже вариант с таким ID

if FindPCBuildOption(List, Data.ID) <> nil then

Exit;

// Создаем новый узел

New(NewNode);

NewNode^.Data := Data;

NewNode^.Next := List.Head;

// Добавляем в начало списка

List.Head := NewNode;

Inc(List.Count);

Result := True;

end;

function RemovePCBuildOption(var List: TPCBuildOptionList; ID: Integer): Boolean;

var

Current, Previous: PPCBuildOptionNode;

begin

Result := False;

if List.Head = nil then

Exit;

// Если удаляемый элемент - первый в списке

if List.Head^.Data.ID = ID then

begin

Current := List.Head;

List.Head := List.Head^.Next;

Dispose(Current);

Dec(List.Count);

Result := True;

Exit;

end;

// Ищем элемент в списке

Previous := List.Head;

Current := List.Head^.Next;

while (Current <> nil) and (Current^.Data.ID <> ID) do

begin

Previous := Current;

Current := Current^.Next;

end;

// Если элемент найден

if Current <> nil then

begin

Previous^.Next := Current^.Next;

Dispose(Current);

Dec(List.Count);

Result := True;

end;

end;

function FindPCBuildOption(const List: TPCBuildOptionList; ID: Integer): PPCBuildOptionNode;

var

Current: PPCBuildOptionNode;

begin

Result := nil;

Current := List.Head;

while Current <> nil do

begin

if Current^.Data.ID = ID then

begin

Result := Current;

Exit;

end;

Current := Current^.Next;

end;

end;

procedure ClearPCBuildOptionList(var List: TPCBuildOptionList);

var

Current, Next: PPCBuildOptionNode;

begin

Current := List.Head;

while Current <> nil do

begin

Next := Current^.Next;

Dispose(Current);

Current := Next;

end;

List.Head := nil;

List.Count := 0;

end;

{ Реализация функций для работы со списком заказов }

procedure InitOrderList(var List: TOrderList);

begin

List.Head := nil;

List.Count := 0;

end;

function AddOrder(var List: TOrderList; const Data: TOrder): Boolean;

var

NewNode: POrderNode;

begin

Result := False;

// Проверяем, не существует ли уже заказ с таким ID

if FindOrder(List, Data.ID) <> nil then

Exit;

// Создаем новый узел

New(NewNode);

NewNode^.Data := Data;

NewNode^.Next := List.Head;

// Добавляем в начало списка

List.Head := NewNode;

Inc(List.Count);

Result := True;

end;

function RemoveOrder(var List: TOrderList; ID: Integer): Boolean;

var

Current, Previous: POrderNode;

begin

Result := False;

if List.Head = nil then

Exit;

// Если удаляемый элемент - первый в списке

if List.Head^.Data.ID = ID then

begin

Current := List.Head;

List.Head := List.Head^.Next;

Dispose(Current);

Dec(List.Count);

Result := True;

Exit;

end;

// Ищем элемент в списке

Previous := List.Head;

Current := List.Head^.Next;

while (Current <> nil) and (Current^.Data.ID <> ID) do

begin

Previous := Current;

Current := Current^.Next;

end;

// Если элемент найден

if Current <> nil then

begin

Previous^.Next := Current^.Next;

Dispose(Current);

Dec(List.Count);

Result := True;

end;

end;

function FindOrder(const List: TOrderList; ID: Integer): POrderNode;

var

Current: POrderNode;

begin

Result := nil;

Current := List.Head;

while Current <> nil do

begin

if Current^.Data.ID = ID then

begin

Result := Current;

Exit;

end;

Current := Current^.Next;

end;

end;

procedure ClearOrderList(var List: TOrderList);

var

Current, Next: POrderNode;

begin

Current := List.Head;

while Current <> nil do

begin

Next := Current^.Next;

Dispose(Current);

Current := Next;

end;

List.Head := nil;

List.Count := 0;

end;

end.

======== FileOperations.pas ========

unit FileOperations;

{$mode objfpc}{$H+}

interface

uses

SysUtils, DataTypes, DynamicLists;

const

COMPONENTS\_FILE = 'data/components.dat';

COMPONENT\_TYPES\_FILE = 'data/component\_types.dat';

COMPATIBILITY\_FILE = 'data/compatibility.dat';

ORDERS\_FILE = 'data/orders.dat';

{ Функции для работы с файлом типов комплектующих }

function LoadComponentTypes(var List: TComponentTypeList; const FileName: string = COMPONENT\_TYPES\_FILE): Boolean;

function SaveComponentTypes(const List: TComponentTypeList; const FileName: string = COMPONENT\_TYPES\_FILE): Boolean;

{ Функции для работы с файлом комплектующих }

function LoadComponents(var List: TComponentList; const FileName: string = COMPONENTS\_FILE): Boolean;

function SaveComponents(const List: TComponentList; const FileName: string = COMPONENTS\_FILE): Boolean;

{ Функции для работы с файлом совместимости }

function LoadCompatibility(var List: TCompatibilityList; const FileName: string = COMPATIBILITY\_FILE): Boolean;

function SaveCompatibility(const List: TCompatibilityList; const FileName: string = COMPATIBILITY\_FILE): Boolean;

{ Функции для работы с файлом заказов }

function LoadOrders(var List: TOrderList; const FileName: string = ORDERS\_FILE): Boolean;

function SaveOrders(const List: TOrderList; const FileName: string = ORDERS\_FILE): Boolean;

{ Функции для сохранения результатов специальных функций в текстовые файлы }

function SavePCBuildOptionsToTextFile(const List: TPCBuildOptionList;

const ComponentList: TComponentList;

const FileName: string = 'data/pc\_build\_options.txt'): Boolean;

function SaveOrderToTextFile(const Order: TOrder;

const BuildOption: TPCBuildOption;

const ComponentList: TComponentList;

const FileName: string = ''): Boolean;

implementation

{ Реализация функций для работы с файлом типов комплектующих }

function LoadComponentTypes(var List: TComponentTypeList; const FileName: string): Boolean;

var

F: file of TComponentType;

Item: TComponentType;

begin

Result := False;

// Очищаем список перед загрузкой

ClearComponentTypeList(List);

// Проверяем существование файла

if not FileExists(FileName) then

Exit;

try

// Открываем файл для чтения

AssignFile(F, FileName);

Reset(F);

// Читаем записи из файла и добавляем их в список

while not Eof(F) do

begin

Read(F, Item);

// Преобразуем фиксированную строку в обычную

Item.Name := FixedToString(Item.Name);

AddComponentType(List, Item);

end;

// Закрываем файл

CloseFile(F);

Result := True;

except

// В случае ошибки закрываем файл и возвращаем False

if IOResult = 0 then

CloseFile(F);

Result := False;

end;

end;

function SaveComponentTypes(const List: TComponentTypeList; const FileName: string): Boolean;

var

F: file of TComponentType;

Current: PComponentTypeNode;

TempData: TComponentType;

begin

Result := False;

try

// Открываем файл для записи

AssignFile(F, FileName);

Rewrite(F);

// Проходим по списку и записываем каждый элемент в файл

Current := List.Head;

while Current <> nil do

begin

// Создаем временную копию с фиксированными строками

TempData.TypeCode := Current^.Data.TypeCode;

TempData.Name := StringToFixed(Current^.Data.Name);

Write(F, TempData);

Current := Current^.Next;

end;

// Закрываем файл

CloseFile(F);

Result := True;

except

// В случае ошибки закрываем файл и возвращаем False

if IOResult = 0 then

CloseFile(F);

Result := False;

end;

end;

{ Реализация функций для работы с файлом комплектующих }

function LoadComponents(var List: TComponentList; const FileName: string): Boolean;

var

F: file of TComponent;

Item: TComponent;

begin

Result := False;

// Очищаем список перед загрузкой

ClearComponentList(List);

// Проверяем существование файла

if not FileExists(FileName) then

Exit;

try

// Открываем файл для чтения

AssignFile(F, FileName);

Reset(F);

// Читаем записи из файла и добавляем их в список

while not Eof(F) do

begin

Read(F, Item);

// Преобразуем фиксированные строки в обычные

Item.Manufacturer := FixedToString(Item.Manufacturer);

Item.Model := FixedToString(Item.Model);

Item.Parameters := FixedToString(Item.Parameters);

AddComponent(List, Item);

end;

// Закрываем файл

CloseFile(F);

Result := True;

except

// В случае ошибки закрываем файл и возвращаем False

if IOResult = 0 then

CloseFile(F);

Result := False;

end;

end;

function SaveComponents(const List: TComponentList; const FileName: string): Boolean;

var

F: file of TComponent;

Current: PComponentNode;

TempData: TComponent;

begin

Result := False;

try

// Открываем файл для записи

AssignFile(F, FileName);

Rewrite(F);

// Проходим по списку и записываем каждый элемент в файл

Current := List.Head;

while Current <> nil do

begin

// Создаем временную копию с фиксированными строками

TempData.Code := Current^.Data.Code;

TempData.TypeCode := Current^.Data.TypeCode;

TempData.Manufacturer := StringToFixed(Current^.Data.Manufacturer);

TempData.Model := StringToFixed(Current^.Data.Model);

TempData.Parameters := StringToFixed(Current^.Data.Parameters);

TempData.Price := Current^.Data.Price;

TempData.InStock := Current^.Data.InStock;

Write(F, TempData);

Current := Current^.Next;

end;

// Закрываем файл

CloseFile(F);

Result := True;

except

// В случае ошибки закрываем файл и возвращаем False

if IOResult = 0 then

CloseFile(F);

Result := False;

end;

end;

{ Реализация функций для работы с файлом совместимости }

function LoadCompatibility(var List: TCompatibilityList; const FileName: string): Boolean;

var

F: file of TCompatibility;

Item: TCompatibility;

begin

Result := False;

// Очищаем список перед загрузкой

ClearCompatibilityList(List);

// Проверяем существование файла

if not FileExists(FileName) then

Exit;

try

// Открываем файл для чтения

AssignFile(F, FileName);

Reset(F);

// Читаем записи из файла и добавляем их в список

while not Eof(F) do

begin

Read(F, Item);

AddCompatibility(List, Item);

end;

// Закрываем файл

CloseFile(F);

Result := True;

except

// В случае ошибки закрываем файл и возвращаем False

if IOResult = 0 then

CloseFile(F);

Result := False;

end;

end;

function SaveCompatibility(const List: TCompatibilityList; const FileName: string): Boolean;

var

F: file of TCompatibility;

Current: PCompatibilityNode;

begin

Result := False;

try

// Открываем файл для записи

AssignFile(F, FileName);

Rewrite(F);

// Проходим по списку и записываем каждый элемент в файл

Current := List.Head;

while Current <> nil do

begin

Write(F, Current^.Data);

Current := Current^.Next;

end;

// Закрываем файл

CloseFile(F);

Result := True;

except

// В случае ошибки закрываем файл и возвращаем False

if IOResult = 0 then

CloseFile(F);

Result := False;

end;

end;

{ Реализация функций для работы с файлом заказов }

function LoadOrders(var List: TOrderList; const FileName: string): Boolean;

var

F: file of TOrder;

Item: TOrder;

begin

Result := False;

// Очищаем список перед загрузкой

ClearOrderList(List);

// Проверяем существование файла

if not FileExists(FileName) then

Exit;

try

// Открываем файл для чтения

AssignFile(F, FileName);

Reset(F);

// Читаем записи из файла и добавляем их в список

while not Eof(F) do

begin

Read(F, Item);

// Преобразуем фиксированные строки в обычные

Item.CustomerName := FixedToString(Item.CustomerName);

Item.CustomerPhone := FixedToString(Item.CustomerPhone);

AddOrder(List, Item);

end;

// Закрываем файл

CloseFile(F);

Result := True;

except

// В случае ошибки закрываем файл и возвращаем False

if IOResult = 0 then

CloseFile(F);

Result := False;

end;

end;

function SaveOrders(const List: TOrderList; const FileName: string): Boolean;

var

F: file of TOrder;

Current: POrderNode;

TempData: TOrder;

begin

Result := False;

try

// Открываем файл для записи

AssignFile(F, FileName);

Rewrite(F);

// Проходим по списку и записываем каждый элемент в файл

Current := List.Head;

while Current <> nil do

begin

// Создаем временную копию с фиксированными строками

TempData.ID := Current^.Data.ID;

TempData.BuildOptionID := Current^.Data.BuildOptionID;

TempData.CustomerName := StringToFixed(Current^.Data.CustomerName);

TempData.CustomerPhone := StringToFixed(Current^.Data.CustomerPhone);

TempData.OrderDate := Current^.Data.OrderDate;

Write(F, TempData);

Current := Current^.Next;

end;

// Закрываем файл

CloseFile(F);

Result := True;

except

// В случае ошибки закрываем файл и возвращаем False

if IOResult = 0 then

CloseFile(F);

Result := False;

end;

end;

{ Реализация функций для сохранения результатов специальных функций в текстовые файлы }

function SavePCBuildOptionsToTextFile(const List: TPCBuildOptionList;

const ComponentList: TComponentList;

const FileName: string): Boolean;

var

F: TextFile;

Current: PPCBuildOptionNode;

ComponentNode: PComponentNode;

i: Integer;

ComponentCode: Integer;

begin

Result := False;

try

// Открываем текстовый файл для записи

AssignFile(F, FileName);

Rewrite(F);

// Записываем заголовок

WriteLn(F, 'Варианты комплектации ПК');

WriteLn(F, '════════════════════════');

WriteLn(F);

// Проходим по списку вариантов комплектации

Current := List.Head;

while Current <> nil do

begin

WriteLn(F, 'Вариант #', Current^.Data.ID);

WriteLn(F, 'Общая стоимость: ', Current^.Data.TotalPrice:0:2, ' руб.');

WriteLn(F, 'Комплектующие:');

// Выводим информацию о каждой комплектующей в этом варианте

for i := 0 to Length(Current^.Data.ComponentCodes) - 1 do

begin

ComponentCode := Current^.Data.ComponentCodes[i];

ComponentNode := FindComponent(ComponentList, ComponentCode);

if ComponentNode <> nil then

begin

WriteLn(F, ' • ', ComponentNode^.Data.Manufacturer, ' ',

ComponentNode^.Data.Model, ' (',

ComponentNode^.Data.Price:0:2, ' руб.)');

end;

end;

WriteLn(F, '------------------------');

WriteLn(F);

Current := Current^.Next;

end;

// Закрываем файл

CloseFile(F);

Result := True;

except

// В случае ошибки закрываем файл и возвращаем False

if IOResult = 0 then

CloseFile(F);

Result := False;

end;

end;

function SaveOrderToTextFile(const Order: TOrder;

const BuildOption: TPCBuildOption;

const ComponentList: TComponentList;

const FileName: string): Boolean;

var

F: TextFile;

ComponentNode: PComponentNode;

i: Integer;

ComponentCode: Integer;

ActualFileName: string;

begin

Result := False;

// Если имя файла не указано, генерируем его на основе ID заказа

if FileName = '' then

ActualFileName := 'data/order\_confirmation\_' + IntToStr(Order.ID) + '.txt'

else

ActualFileName := FileName;

try

// Открываем текстовый файл для записи

AssignFile(F, ActualFileName);

Rewrite(F);

// Записываем информацию о заказе

WriteLn(F, 'Подтверждение заказа #', Order.ID);

WriteLn(F, '═══════════════════════════════');

WriteLn(F);

WriteLn(F, 'Дата заказа: ', FormatDateTime('dd.mm.yyyy hh:nn', Order.OrderDate));

WriteLn(F, 'Заказчик: ', Order.CustomerName);

WriteLn(F, 'Телефон: ', Order.CustomerPhone);

WriteLn(F, 'Общая стоимость: ', BuildOption.TotalPrice:0:2, ' руб.');

WriteLn(F);

WriteLn(F, 'Комплектующие:');

// Выводим информацию о каждой комплектующей в заказе

for i := 0 to Length(BuildOption.ComponentCodes) - 1 do

begin

ComponentCode := BuildOption.ComponentCodes[i];

ComponentNode := FindComponent(ComponentList, ComponentCode);

if ComponentNode <> nil then

begin

WriteLn(F, ' • ', ComponentNode^.Data.Manufacturer, ' ',

ComponentNode^.Data.Model, ' (',

ComponentNode^.Data.Price:0:2, ' руб.)');

end;

end;

WriteLn(F);

WriteLn(F, 'Спасибо за ваш заказ!');

// Закрываем файл

CloseFile(F);

Result := True;

except

// В случае ошибки закрываем файл и возвращаем False

if IOResult = 0 then

CloseFile(F);

Result := False;

end;

end;

end.

======== InputUtils.pas ========

unit InputUtils;

{$mode objfpc}{$H+}

interface

uses

SysUtils;

const

// Константы для специальных команд ввода

CMD\_RETURN\_TO\_MENU = '/menu';

CMD\_CANCEL = '/cancel';

// Функции для безопасного чтения различных типов данных

function SafeReadInteger(const Prompt: string; MinValue: Integer = Low(Integer); MaxValue: Integer = High(Integer);

const ErrorMsg: string = ''): Integer;

function SafeReadFloat(const Prompt: string; MinValue: Real = -1E38; MaxValue: Real = 1E38;

const ErrorMsg: string = ''): Real;

function SafeReadString(const Prompt: string; MinLength: Integer = 0; MaxLength: Integer = High(Integer);

const ErrorMsg: string = ''): string;

function SafeReadBoolean(const Prompt: string; const ErrorMsg: string = ''): Boolean;

// Функция для проверки, хочет ли пользователь вернуться в меню

function IsReturnToMenuCommand(const Input: string): Boolean;

// Функция для проверки, хочет ли пользователь отменить текущую операцию

function IsCancelCommand(const Input: string): Boolean;

// Функция для безопасного подтверждения операций

function SafeReadConfirmation(const Prompt: string; const ErrorMsg: string = ''): Boolean;

implementation

function SafeReadInteger(const Prompt: string; MinValue: Integer; MaxValue: Integer; const ErrorMsg: string): Integer;

var

InputStr: string;

Code: Integer;

DefaultErrorMsg: string;

begin

DefaultErrorMsg := Format('Пожалуйста, введите целое число от %d до %d или "%s" для возврата в меню',

[MinValue, MaxValue, CMD\_RETURN\_TO\_MENU]);

if ErrorMsg <> '' then

DefaultErrorMsg := ErrorMsg;

repeat

Write(Prompt);

ReadLn(InputStr);

// Проверка на команду возврата в меню

if IsReturnToMenuCommand(InputStr) then

begin

Result := MinValue - 1; // Специальное значение для индикации возврата в меню

Exit;

end;

// Проверка на команду отмены

if IsCancelCommand(InputStr) then

begin

Result := MinValue - 2; // Специальное значение для индикации отмены

Exit;

end;

// Попытка преобразовать строку в целое число

Val(InputStr, Result, Code);

if (Code <> 0) or (Result < MinValue) or (Result > MaxValue) then

begin

WriteLn(DefaultErrorMsg);

Code := 1; // Установка кода ошибки для повторного запроса

end;

until Code = 0;

end;

function SafeReadFloat(const Prompt: string; MinValue: Real; MaxValue: Real; const ErrorMsg: string): Real;

var

InputStr: string;

Code: Integer;

DefaultErrorMsg: string;

begin

DefaultErrorMsg := Format('Пожалуйста, введите число от %f до %f или "%s" для возврата в меню',

[MinValue, MaxValue, CMD\_RETURN\_TO\_MENU]);

if ErrorMsg <> '' then

DefaultErrorMsg := ErrorMsg;

repeat

Write(Prompt);

ReadLn(InputStr);

// Проверка на команду возврата в меню

if IsReturnToMenuCommand(InputStr) then

begin

Result := MinValue - 1; // Специальное значение для индикации возврата в меню

Exit;

end;

// Проверка на команду отмены

if IsCancelCommand(InputStr) then

begin

Result := MinValue - 2; // Специальное значение для индикации отмены

Exit;

end;

// Попытка преобразовать строку в число с плавающей точкой

Val(InputStr, Result, Code);

if (Code <> 0) or (Result < MinValue) or (Result > MaxValue) then

begin

WriteLn(DefaultErrorMsg);

Code := 1; // Установка кода ошибки для повторного запроса

end;

until Code = 0;

end;

function SafeReadString(const Prompt: string; MinLength: Integer; MaxLength: Integer; const ErrorMsg: string): string;

var

InputStr: string;

DefaultErrorMsg: string;

begin

DefaultErrorMsg := Format('Пожалуйста, введите строку длиной от %d до %d символов или "%s" для возврата в меню',

[MinLength, MaxLength, CMD\_RETURN\_TO\_MENU]);

if ErrorMsg <> '' then

DefaultErrorMsg := ErrorMsg;

repeat

Write(Prompt);

ReadLn(InputStr);

// Проверка на команду возврата в меню

if IsReturnToMenuCommand(InputStr) then

begin

Result := CMD\_RETURN\_TO\_MENU;

Exit;

end;

// Проверка на команду отмены

if IsCancelCommand(InputStr) then

begin

Result := CMD\_CANCEL;

Exit;

end;

// Проверка длины строки

if (Length(InputStr) < MinLength) or (Length(InputStr) > MaxLength) then

begin

WriteLn(DefaultErrorMsg);

Continue;

end;

Result := InputStr;

Break;

until False;

end;

function SafeReadBoolean(const Prompt: string; const ErrorMsg: string): Boolean;

var

InputStr: string;

DefaultErrorMsg: string;

begin

DefaultErrorMsg := Format('Пожалуйста, введите "да", "нет" или "%s" для возврата в меню', [CMD\_RETURN\_TO\_MENU]);

if ErrorMsg <> '' then

DefaultErrorMsg := ErrorMsg;

repeat

Write(Prompt);

ReadLn(InputStr);

// Проверка на команду возврата в меню

if IsReturnToMenuCommand(InputStr) then

begin

Result := False; // Специальное значение для индикации возврата в меню

Exit;

end;

// Проверка на команду отмены

if IsCancelCommand(InputStr) then

begin

Result := False; // Специальное значение для индикации отмены

Exit;

end;

// Проверка на "да"

if (InputStr = 'да') or (InputStr = 'Да') or (InputStr = 'ДА') or

(InputStr = 'y') or (InputStr = 'Y') or (InputStr = 'yes') or (InputStr = 'Yes') then

begin

Result := True;

Exit;

end;

// Проверка на "нет"

if (InputStr = 'нет') or (InputStr = 'Нет') or (InputStr = 'НЕТ') or

(InputStr = 'n') or (InputStr = 'N') or (InputStr = 'no') or (InputStr = 'No') then

begin

Result := False;

Exit;

end;

WriteLn(DefaultErrorMsg);

until False;

end;

function IsReturnToMenuCommand(const Input: string): Boolean;

begin

Result := (Input = CMD\_RETURN\_TO\_MENU);

end;

function IsCancelCommand(const Input: string): Boolean;

begin

Result := (Input = CMD\_CANCEL);

end;

function SafeReadConfirmation(const Prompt: string; const ErrorMsg: string = ''): Boolean;

var

InputStr: string;

DefaultErrorMsg: string;

begin

DefaultErrorMsg := Format('Пожалуйста, введите "да", "нет" или "%s" для возврата в меню', [CMD\_RETURN\_TO\_MENU]);

if ErrorMsg <> '' then

DefaultErrorMsg := ErrorMsg;

repeat

Write(Prompt);

ReadLn(InputStr);

// Проверка на команду возврата в меню

if IsReturnToMenuCommand(InputStr) then

begin

Result := False; // Специальное значение для индикации возврата в меню

Exit;

end;

// Проверка на команду отмены

if IsCancelCommand(InputStr) then

begin

Result := False; // Специальное значение для индикации отмены

Exit;

end;

// Проверка на "да"

if (InputStr = 'да') or (InputStr = 'Да') or (InputStr = 'ДА') or

(InputStr = 'y') or (InputStr = 'Y') or (InputStr = 'yes') or (InputStr = 'Yes') then

begin

Result := True;

Exit;

end;

// Проверка на "нет"

if (InputStr = 'нет') or (InputStr = 'Нет') or (InputStr = 'НЕТ') or

(InputStr = 'n') or (InputStr = 'N') or (InputStr = 'no') or (InputStr = 'No') then

begin

Result := False;

Exit;

end;

WriteLn(DefaultErrorMsg);

until False;

end;

end.

======== MainProgram.pas ========

unit MainProgram;

{$mode objfpc}{$H+}

interface

uses

SysUtils,

DataTypes,

DynamicLists,

UI,

InputUtils,

MenuHandlers,

DataSorting,

DataSearch;

// Процедуры для обработки различных пунктов меню

procedure HandleSortData(var ComponentTypesList: TComponentTypeList;

var ComponentsList: TComponentList;

var OrdersList: TOrderList);

procedure HandleSearchData(const ComponentsList: TComponentList;

const ComponentTypesList: TComponentTypeList);

// Процедура для инициализации всех списков

procedure InitAllLists(var ComponentTypesList: TComponentTypeList;

var ComponentsList: TComponentList;

var CompatibilityList: TCompatibilityList;

var PCBuildOptionsList: TPCBuildOptionList;

var OrdersList: TOrderList);

implementation

// Процедура для инициализации списков

procedure InitAllLists(var ComponentTypesList: TComponentTypeList;

var ComponentsList: TComponentList;

var CompatibilityList: TCompatibilityList;

var PCBuildOptionsList: TPCBuildOptionList;

var OrdersList: TOrderList);

begin

InitComponentTypeList(ComponentTypesList);

InitComponentList(ComponentsList);

InitCompatibilityList(CompatibilityList);

InitPCBuildOptionList(PCBuildOptionsList);

InitOrderList(OrdersList);

end;

// Процедура для сортировки данных

procedure HandleSortData(var ComponentTypesList: TComponentTypeList;

var ComponentsList: TComponentList;

var OrdersList: TOrderList);

var

SortChoice: Integer;

begin

WriteLn('=== СОРТИРОВКА ДАННЫХ ===');

WriteLn('Выберите тип данных для сортировки:');

WriteLn('1. Типы комплектующих (по коду)');

WriteLn('2. Комплектующие (по цене)');

WriteLn('3. Комплектующие (по коду)');

WriteLn('4. Комплектующие (по производителю)');

WriteLn('5. Заказы (по дате)');

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

SortChoice := SafeReadInteger('Выберите пункт (1-5): ', 1, 5, 'Ошибка: Выберите пункт от 1 до 5.');

case SortChoice of

1: SortComponentTypesByCode(ComponentTypesList);

2: SortComponentsByPrice(ComponentsList);

3: SortComponentsByCode(ComponentsList);

4: SortComponentsByManufacturer(ComponentsList);

5: SortOrdersByDate(OrdersList);

end;

WriteLn('Данные отсортированы.');

PressEnterToContinue;

end;

// Процедура для поиска данных с фильтрами

procedure HandleSearchData(const ComponentsList: TComponentList;

const ComponentTypesList: TComponentTypeList);

var

SearchChoice: Integer;

SearchResults: TComponentList;

FilterMinPrice, FilterMaxPrice: Real;

FilterManufacturer, FilterModel: string;

FilterTypeCode: Integer;

FilterMinInStock: Integer;

begin

WriteLn('=== ПОИСК ДАННЫХ С ФИЛЬТРАМИ ===');

WriteLn('Выберите тип поиска:');

WriteLn('1. Комплектующие по диапазону цен');

WriteLn('2. Комплектующие по производителю');

WriteLn('3. Комплектующие по типу');

WriteLn('4. Комплектующие по модели');

WriteLn('5. Комплектующие с минимальным количеством в наличии');

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

SearchChoice := SafeReadInteger('Выберите пункт (1-5): ', 1, 5, 'Ошибка: Выберите пункт от 1 до 5.');

InitComponentList(SearchResults);

case SearchChoice of

1: begin

FilterMinPrice := SafeReadFloat('Введите минимальную цену: ', 0, 1E10);

FilterMaxPrice := SafeReadFloat('Введите максимальную цену: ', FilterMinPrice, 1E10);

SearchComponentsByPriceRange(ComponentsList, FilterMinPrice, FilterMaxPrice, SearchResults);

end;

2: begin

FilterManufacturer := SafeReadString('Введите производителя: ', 1, 50);

SearchComponentsByManufacturer(ComponentsList, FilterManufacturer, SearchResults);

end;

3: begin

FilterTypeCode := SafeReadInteger('Введите код типа комплектующей: ', 1, High(Integer));

SearchComponentsByType(ComponentsList, FilterTypeCode, SearchResults);

end;

4: begin

FilterModel := SafeReadString('Введите модель: ', 1, 50);

SearchComponentsByModel(ComponentsList, FilterModel, SearchResults);

end;

5: begin

FilterMinInStock := SafeReadInteger('Введите минимальное количество в наличии: ', 0, High(Integer));

SearchComponentsByMinStock(ComponentsList, FilterMinInStock, SearchResults);

end;

end;

WriteLn;

WriteLn('=== РЕЗУЛЬТАТЫ ПОИСКА ===');

if SearchResults.Count > 0 then

begin

WriteLn('Найдено записей: ', SearchResults.Count);

DisplayComponents(SearchResults, ComponentTypesList);

end

else

begin

WriteLn('По заданным критериям ничего не найдено.');

PressEnterToContinue;

end;

ClearComponentList(SearchResults);

end;

end.

======== MenuHandlers.pas ========

unit MenuHandlers;

{$mode objfpc}{$H+}

interface

uses

SysUtils,

DataTypes,

DynamicLists,

FileOperations,

UI,

SpecialFunctions,

InputUtils;

// Процедуры для обработки пунктов меню

procedure LoadAllData(var ComponentTypesList: TComponentTypeList;

var ComponentsList: TComponentList;

var CompatibilityList: TCompatibilityList;

var OrdersList: TOrderList);

procedure SaveAllData(const ComponentTypesList: TComponentTypeList;

const ComponentsList: TComponentList;

const CompatibilityList: TCompatibilityList;

const OrdersList: TOrderList);

procedure ViewList(const ComponentTypesList: TComponentTypeList;

const ComponentsList: TComponentList;

const CompatibilityList: TCompatibilityList;

const OrdersList: TOrderList;

const PCBuildOptionsList: TPCBuildOptionList);

procedure AddData(var ComponentTypesList: TComponentTypeList;

var ComponentsList: TComponentList;

var CompatibilityList: TCompatibilityList);

procedure RemoveData(var ComponentTypesList: TComponentTypeList;

var ComponentsList: TComponentList;

var CompatibilityList: TCompatibilityList);

procedure EditData(var ComponentTypesList: TComponentTypeList;

var ComponentsList: TComponentList;

var CompatibilityList: TCompatibilityList);

procedure HandleSpecialFunctions(const ComponentTypesList: TComponentTypeList;

const ComponentsList: TComponentList;

const CompatibilityList: TCompatibilityList;

var PCBuildOptionsList: TPCBuildOptionList;

var OrdersList: TOrderList);

implementation

// Процедуры для инициализации списков

procedure InitAllLists(var ComponentTypesList: TComponentTypeList;

var ComponentsList: TComponentList;

var CompatibilityList: TCompatibilityList;

var PCBuildOptionsList: TPCBuildOptionList;

var OrdersList: TOrderList);

begin

InitComponentTypeList(ComponentTypesList);

InitComponentList(ComponentsList);

InitCompatibilityList(CompatibilityList);

InitPCBuildOptionList(PCBuildOptionsList);

InitOrderList(OrdersList);

end;

// Процедура для загрузки данных из файлов

procedure LoadAllData(var ComponentTypesList: TComponentTypeList;

var ComponentsList: TComponentList;

var CompatibilityList: TCompatibilityList;

var OrdersList: TOrderList);

var

Confirmation: string;

PCBuildOptionsList: TPCBuildOptionList;

begin

// Запрос подтверждения у пользователя

WriteLn('ВНИМАНИЕ: Вы собираетесь загрузить данные из файлов.');

WriteLn('Это действие заменит все текущие данные в памяти.');

Write('Вы уверены, что хотите продолжить? (да/нет): ');

ReadLn(Confirmation);

if (Confirmation <> 'да') and (Confirmation <> 'Да') and (Confirmation <> 'ДА') then

begin

WriteLn('Загрузка данных отменена.');

PressEnterToContinue;

Exit;

end;

// Инициализация списков

InitAllLists(ComponentTypesList, ComponentsList, CompatibilityList, PCBuildOptionsList, OrdersList);

// Загрузка данных из файлов

if not LoadComponentTypes(ComponentTypesList) then

WriteLn('Файл типов комплектующих не найден или поврежден. Создан пустой список.');

if not LoadComponents(ComponentsList) then

WriteLn('Файл комплектующих не найден или поврежден. Создан пустой список.');

if not LoadCompatibility(CompatibilityList) then

WriteLn('Файл совместимости не найден или поврежден. Создан пустой список.');

if not LoadOrders(OrdersList) then

WriteLn('Файл заказов не найден или поврежден. Создан пустой список.');

WriteLn('Данные загружены.');

PressEnterToContinue;

end;

// Процедура для сохранения данных в файлы

procedure SaveAllData(const ComponentTypesList: TComponentTypeList;

const ComponentsList: TComponentList;

const CompatibilityList: TCompatibilityList;

const OrdersList: TOrderList);

begin

if not SaveComponentTypes(ComponentTypesList) then

WriteLn('Ошибка при сохранении типов комплектующих.');

if not SaveComponents(ComponentsList) then

WriteLn('Ошибка при сохранении комплектующих.');

if not SaveCompatibility(CompatibilityList) then

WriteLn('Ошибка при сохранении данных о совместимости.');

if not SaveOrders(OrdersList) then

WriteLn('Ошибка при сохранении заказов.');

WriteLn('Данные сохранены.');

end;

// Процедура для обработки пункта меню "Просмотр всего списка"

procedure ViewList(const ComponentTypesList: TComponentTypeList;

const ComponentsList: TComponentList;

const CompatibilityList: TCompatibilityList;

const OrdersList: TOrderList;

const PCBuildOptionsList: TPCBuildOptionList);

var

SubmenuChoice: Integer;

begin

DisplayListSubmenu;

SubmenuChoice := GetListSubmenuChoice;

case SubmenuChoice of

SUBMENU\_COMPONENT\_TYPES:

DisplayComponentTypes(ComponentTypesList);

SUBMENU\_COMPONENTS:

DisplayComponents(ComponentsList, ComponentTypesList);

SUBMENU\_COMPATIBILITY:

DisplayCompatibility(CompatibilityList, ComponentsList);

SUBMENU\_ORDERS:

DisplayOrders(OrdersList, PCBuildOptionsList);

end;

end;

// Процедура для обработки пункта меню "Добавление данных в список"

procedure AddData(var ComponentTypesList: TComponentTypeList;

var ComponentsList: TComponentList;

var CompatibilityList: TCompatibilityList);

var

SubmenuChoice: Integer;

NewComponentType: TComponentType;

NewComponent: TComponent;

NewCompatibility: TCompatibility;

Confirmation: string;

begin

DisplayAddDataSubmenu;

SubmenuChoice := GetAddDataSubmenuChoice;

// Проверка на отмену операции

if SubmenuChoice < SUBMENU\_COMPONENT\_TYPES then

begin

WriteLn('Операция добавления отменена.');

PressEnterToContinue;

Exit;

end;

WriteLn('Для прерывания операции в любой момент введите "/cancel" или "/menu"');

WriteLn;

case SubmenuChoice of

SUBMENU\_COMPONENT\_TYPES:

begin

WriteLn('=== ДОБАВЛЕНИЕ ТИПА КОМПЛЕКТУЮЩЕЙ ===');

if InputComponentType(NewComponentType, ComponentTypesList) then

begin

if AddComponentType(ComponentTypesList, NewComponentType) then

WriteLn('Тип комплектующей успешно добавлен.')

else

WriteLn('Ошибка: Тип с таким кодом уже существует.');

end

else

WriteLn('Добавление типа комплектующей отменено.');

end;

SUBMENU\_COMPONENTS:

begin

WriteLn('=== ДОБАВЛЕНИЕ КОМПЛЕКТУЮЩЕЙ ===');

if ComponentTypesList.Count = 0 then

begin

WriteLn('Ошибка: Сначала необходимо добавить типы комплектующих.');

PressEnterToContinue;

Exit;

end;

WriteLn('Доступные типы комплектующих:');

DisplayComponentTypes(ComponentTypesList);

if InputComponent(NewComponent, ComponentTypesList, ComponentsList) then

begin

if AddComponent(ComponentsList, NewComponent) then

WriteLn('Комплектующая успешно добавлена.')

else

WriteLn('Ошибка: Комплектующая с таким кодом уже существует.');

end

else

WriteLn('Добавление комплектующей отменено.');

end;

SUBMENU\_COMPATIBILITY:

begin

WriteLn('=== ДОБАВЛЕНИЕ ЗАПИСИ О СОВМЕСТИМОСТИ ===');

if ComponentsList.Count < 2 then

begin

WriteLn('Ошибка: Для создания записи о совместимости необходимо минимум 2 комплектующих.');

PressEnterToContinue;

Exit;

end;

WriteLn('Доступные комплектующие:');

DisplayComponents(ComponentsList, ComponentTypesList);

if InputCompatibility(NewCompatibility, ComponentsList) then

begin

if NewCompatibility.ComponentCode1 = NewCompatibility.ComponentCode2 then

begin

WriteLn('Ошибка: Нельзя создать запись о совместимости комплектующей с самой собой.');

end

else if AddCompatibility(CompatibilityList, NewCompatibility) then

WriteLn('Запись о совместимости успешно добавлена.')

else

WriteLn('Ошибка: Такая запись о совместимости уже существует.');

end

else

WriteLn('Добавление записи о совместимости отменено.');

end;

end;

PressEnterToContinue;

end;

// Процедура для обработки пункта меню "Удаление данных из списка"

procedure RemoveData(var ComponentTypesList: TComponentTypeList;

var ComponentsList: TComponentList;

var CompatibilityList: TCompatibilityList);

var

SubmenuChoice: Integer;

Code, Code1, Code2: Integer;

ComponentTypeNode: PComponentTypeNode;

ComponentNode: PComponentNode;

HasDependentComponents, HasDependentCompatibility: Boolean;

Current: PComponentNode;

CompatCurrent: PCompatibilityNode;

Confirmation: string;

begin

DisplayListSubmenu;

SubmenuChoice := GetListSubmenuChoice;

// Проверка на отмену операции

if SubmenuChoice < SUBMENU\_COMPONENT\_TYPES then

begin

WriteLn('Операция удаления отменена.');

PressEnterToContinue;

Exit;

end;

WriteLn('Для прерывания операции в любой момент введите "/cancel" или "/menu"');

WriteLn;

case SubmenuChoice of

SUBMENU\_COMPONENT\_TYPES:

begin

WriteLn('=== УДАЛЕНИЕ ТИПА КОМПЛЕКТУЮЩЕЙ ===');

if ComponentTypesList.Count = 0 then

begin

WriteLn('Список типов комплектующих пуст.');

PressEnterToContinue;

Exit;

end;

WriteLn('Доступные типы комплектующих:');

DisplayComponentTypes(ComponentTypesList);

Code := SafeReadInteger('Введите код типа комплектующей для удаления: ', 1);

if Code < 1 then

begin

WriteLn('Удаление типа комплектующей отменено.');

PressEnterToContinue;

Exit;

end;

ComponentTypeNode := FindComponentType(ComponentTypesList, Code);

if ComponentTypeNode = nil then

begin

WriteLn('Ошибка: Тип с таким кодом не найден.');

PressEnterToContinue;

Exit;

end;

// Проверяем, есть ли комплектующие данного типа

HasDependentComponents := False;

Current := ComponentsList.Head;

while Current <> nil do

begin

if Current^.Data.TypeCode = Code then

begin

HasDependentComponents := True;

Break;

end;

Current := Current^.Next;

end;

if HasDependentComponents then

begin

WriteLn('ВНИМАНИЕ: Существуют комплектующие данного типа.');

WriteLn('Удаление типа приведет к проблемам с данными.');

Confirmation := SafeReadString('Вы действительно хотите удалить этот тип? (да/нет): ', 1, 10);

if IsReturnToMenuCommand(Confirmation) or IsCancelCommand(Confirmation) then

begin

WriteLn('Удаление типа комплектующей отменено.');

PressEnterToContinue;

Exit;

end;

if not ((Confirmation = 'да') or (Confirmation = 'Да') or (Confirmation = 'ДА')) then

begin

WriteLn('Удаление типа комплектующей отменено.');

PressEnterToContinue;

Exit;

end;

end;

if RemoveComponentType(ComponentTypesList, Code) then

WriteLn('Тип комплектующей успешно удален.')

else

WriteLn('Ошибка при удалении типа комплектующей.');

end;

SUBMENU\_COMPONENTS:

begin

WriteLn('=== УДАЛЕНИЕ КОМПЛЕКТУЮЩЕЙ ===');

if ComponentsList.Count = 0 then

begin

WriteLn('Список комплектующих пуст.');

PressEnterToContinue;

Exit;

end;

WriteLn('Доступные комплектующие:');

DisplayComponents(ComponentsList, ComponentTypesList);

Code := SafeReadInteger('Введите код комплектующей для удаления: ', 1);

if Code < 1 then

begin

WriteLn('Удаление комплектующей отменено.');

PressEnterToContinue;

Exit;

end;

ComponentNode := FindComponent(ComponentsList, Code);

if ComponentNode = nil then

begin

WriteLn('Ошибка: Комплектующая с таким кодом не найдена.');

PressEnterToContinue;

Exit;

end;

// Проверяем, есть ли записи о совместимости с данной комплектующей

HasDependentCompatibility := False;

CompatCurrent := CompatibilityList.Head;

while CompatCurrent <> nil do

begin

if (CompatCurrent^.Data.ComponentCode1 = Code) or

(CompatCurrent^.Data.ComponentCode2 = Code) then

begin

HasDependentCompatibility := True;

Break;

end;

CompatCurrent := CompatCurrent^.Next;

end;

if HasDependentCompatibility then

begin

WriteLn('ВНИМАНИЕ: Существуют записи о совместимости с данной комплектующей.');

WriteLn('Удаление комплектующей приведет к потере этих записей.');

Confirmation := SafeReadString('Вы действительно хотите удалить эту комплектующую? (да/нет): ', 1, 10);

if IsReturnToMenuCommand(Confirmation) or IsCancelCommand(Confirmation) then

begin

WriteLn('Удаление комплектующей отменено.');

PressEnterToContinue;

Exit;

end;

if not ((Confirmation = 'да') or (Confirmation = 'Да') or (Confirmation = 'ДА')) then

begin

WriteLn('Удаление комплектующей отменено.');

PressEnterToContinue;

Exit;

end;

// Удаляем связанные записи о совместимости

CompatCurrent := CompatibilityList.Head;

while CompatCurrent <> nil do

begin

if (CompatCurrent^.Data.ComponentCode1 = Code) or

(CompatCurrent^.Data.ComponentCode2 = Code) then

begin

RemoveCompatibility(CompatibilityList,

CompatCurrent^.Data.ComponentCode1,

CompatCurrent^.Data.ComponentCode2);

// Начинаем поиск сначала, так как список изменился

CompatCurrent := CompatibilityList.Head;

end

else

CompatCurrent := CompatCurrent^.Next;

end;

WriteLn('Связанные записи о совместимости удалены.');

end;

if RemoveComponent(ComponentsList, Code) then

WriteLn('Комплектующая успешно удалена.')

else

WriteLn('Ошибка при удалении комплектующей.');

end;

SUBMENU\_COMPATIBILITY:

begin

WriteLn('=== УДАЛЕНИЕ ЗАПИСИ О СОВМЕСТИМОСТИ ===');

if CompatibilityList.Count = 0 then

begin

WriteLn('Список записей о совместимости пуст.');

PressEnterToContinue;

Exit;

end;

WriteLn('Доступные записи о совместимости:');

DisplayCompatibility(CompatibilityList, ComponentsList);

Code1 := SafeReadInteger('Введите код первой комплектующей: ', 1);

if Code1 < 1 then

begin

WriteLn('Удаление записи о совместимости отменено.');

PressEnterToContinue;

Exit;

end;

Code2 := SafeReadInteger('Введите код второй комплектующей: ', 1);

if Code2 < 1 then

begin

WriteLn('Удаление записи о совместимости отменено.');

PressEnterToContinue;

Exit;

end;

if FindCompatibility(CompatibilityList, Code1, Code2) = nil then

begin

WriteLn('Ошибка: Запись о совместимости не найдена.');

PressEnterToContinue;

Exit;

end;

if RemoveCompatibility(CompatibilityList, Code1, Code2) then

WriteLn('Запись о совместимости успешно удалена.')

else

WriteLn('Ошибка при удалении записи о совместимости.');

end;

end;

PressEnterToContinue;

end;

// Процедура для обработки пункта меню "Редактирование данных"

procedure EditData(var ComponentTypesList: TComponentTypeList;

var ComponentsList: TComponentList;

var CompatibilityList: TCompatibilityList);

var

SubmenuChoice: Integer;

Code, i, TypeCode: Integer;

ComponentTypeNode: PComponentTypeNode;

ComponentNode: PComponentNode;

CompatibilityNode: PCompatibilityNode;

TempString: string;

TempPrice: Real;

TempStock: Integer;

ComponentCode1, ComponentCode2: Integer;

Component1, Component2: PComponentNode;

begin

DisplayListSubmenu;

SubmenuChoice := GetListSubmenuChoice;

// Проверка на отмену операции

if SubmenuChoice < SUBMENU\_COMPONENT\_TYPES then

begin

WriteLn('Операция редактирования отменена.');

PressEnterToContinue;

Exit;

end;

WriteLn('Для прерывания операции в любой момент введите "/cancel" или "/menu"');

WriteLn;

case SubmenuChoice of

SUBMENU\_COMPONENT\_TYPES:

begin

WriteLn('=== РЕДАКТИРОВАНИЕ ТИПА КОМПЛЕКТУЮЩЕЙ ===');

if ComponentTypesList.Count = 0 then

begin

WriteLn('Список типов комплектующих пуст.');

PressEnterToContinue;

Exit;

end;

WriteLn('Доступные типы комплектующих:');

DisplayComponentTypes(ComponentTypesList);

Code := SafeReadInteger('Введите код типа комплектующей для редактирования: ', 1);

if Code < 1 then

begin

WriteLn('Редактирование типа комплектующей отменено.');

PressEnterToContinue;

Exit;

end;

ComponentTypeNode := FindComponentType(ComponentTypesList, Code);

if ComponentTypeNode = nil then

begin

WriteLn('Ошибка: Тип с таким кодом не найден.');

PressEnterToContinue;

Exit;

end;

WriteLn('Текущие данные:');

WriteLn('Код типа: ', ComponentTypeNode^.Data.TypeCode);

WriteLn('Название: ', ComponentTypeNode^.Data.Name);

WriteLn;

WriteLn('Введите новые данные:');

TempString := SafeReadString('Название типа комплектующей: ', 1, MAX\_STRING\_LENGTH);

if IsReturnToMenuCommand(TempString) or IsCancelCommand(TempString) then

begin

WriteLn('Редактирование типа комплектующей отменено.');

PressEnterToContinue;

Exit;

end;

ComponentTypeNode^.Data.Name := StringToFixed(TempString);

WriteLn('Тип комплектующей успешно отредактирован.');

end;

SUBMENU\_COMPONENTS:

begin

WriteLn('=== РЕДАКТИРОВАНИЕ КОМПЛЕКТУЮЩЕЙ ===');

if ComponentsList.Count = 0 then

begin

WriteLn('Список комплектующих пуст.');

PressEnterToContinue;

Exit;

end;

WriteLn('Доступные комплектующие:');

DisplayComponents(ComponentsList, ComponentTypesList);

Code := SafeReadInteger('Введите код комплектующей для редактирования: ', 1);

if Code < 1 then

begin

WriteLn('Редактирование комплектующей отменено.');

PressEnterToContinue;

Exit;

end;

ComponentNode := FindComponent(ComponentsList, Code);

if ComponentNode = nil then

begin

WriteLn('Ошибка: Комплектующая с таким кодом не найдена.');

PressEnterToContinue;

Exit;

end;

WriteLn('Текущие данные:');

WriteLn('Код: ', ComponentNode^.Data.Code);

WriteLn('Код типа: ', ComponentNode^.Data.TypeCode);

WriteLn('Производитель: ', ComponentNode^.Data.Manufacturer);

WriteLn('Модель: ', ComponentNode^.Data.Model);

WriteLn('Параметры: ', ComponentNode^.Data.Parameters);

WriteLn('Цена: ', ComponentNode^.Data.Price:0:2);

WriteLn('В наличии: ', ComponentNode^.Data.InStock);

WriteLn;

WriteLn('Введите новые данные:');

WriteLn('Доступные типы комплектующих:');

DisplayComponentTypes(ComponentTypesList);

repeat

TypeCode := SafeReadInteger('Код типа комплектующей: ', 1);

if TypeCode < 1 then

begin

WriteLn('Редактирование комплектующей отменено.');

PressEnterToContinue;

Exit;

end;

if FindComponentType(ComponentTypesList, TypeCode) = nil then

WriteLn('Ошибка: Тип с кодом ', TypeCode, ' не найден. Пожалуйста, введите существующий код типа.');

until FindComponentType(ComponentTypesList, TypeCode) <> nil;

ComponentNode^.Data.TypeCode := TypeCode;

TempString := SafeReadString('Производитель: ', 1, MAX\_STRING\_LENGTH);

if IsReturnToMenuCommand(TempString) or IsCancelCommand(TempString) then

begin

WriteLn('Редактирование комплектующей отменено.');

PressEnterToContinue;

Exit;

end;

ComponentNode^.Data.Manufacturer := StringToFixed(TempString);

TempString := SafeReadString('Модель: ', 1, MAX\_STRING\_LENGTH);

if IsReturnToMenuCommand(TempString) or IsCancelCommand(TempString) then

begin

WriteLn('Редактирование комплектующей отменено.');

PressEnterToContinue;

Exit;

end;

ComponentNode^.Data.Model := StringToFixed(TempString);

TempString := SafeReadString('Параметры: ', 0, MAX\_STRING\_LENGTH);

if IsReturnToMenuCommand(TempString) or IsCancelCommand(TempString) then

begin

WriteLn('Редактирование комплектующей отменено.');

PressEnterToContinue;

Exit;

end;

ComponentNode^.Data.Parameters := StringToFixed(TempString);

TempPrice := SafeReadFloat('Цена: ', 0);

if TempPrice < 0 then

begin

WriteLn('Редактирование комплектующей отменено.');

PressEnterToContinue;

Exit;

end;

ComponentNode^.Data.Price := TempPrice;

TempStock := SafeReadInteger('Количество в наличии: ', 0);

if TempStock < 0 then

begin

WriteLn('Редактирование комплектующей отменено.');

PressEnterToContinue;

Exit;

end;

ComponentNode^.Data.InStock := TempStock;

WriteLn('Комплектующая успешно отредактирована.');

end;

SUBMENU\_COMPATIBILITY:

begin

WriteLn('=== РЕДАКТИРОВАНИЕ ЗАПИСИ О СОВМЕСТИМОСТИ ===');

if CompatibilityList.Count = 0 then

begin

WriteLn('Список записей о совместимости пуст.');

PressEnterToContinue;

Exit;

end;

WriteLn('Доступные записи о совместимости:');

DisplayCompatibility(CompatibilityList, ComponentsList);

Code := SafeReadInteger('Введите код первой комплектующей: ', 1);

if Code < 1 then

begin

WriteLn('Редактирование записи о совместимости отменено.');

PressEnterToContinue;

Exit;

end;

i := SafeReadInteger('Введите код второй комплектующей: ', 1);

if i < 1 then

begin

WriteLn('Редактирование записи о совместимости отменено.');

PressEnterToContinue;

Exit;

end;

CompatibilityNode := FindCompatibility(CompatibilityList, Code, i);

if CompatibilityNode = nil then

begin

WriteLn('Ошибка: Запись о совместимости не найдена.');

PressEnterToContinue;

Exit;

end;

WriteLn('Текущие данные:');

WriteLn('Код первой комплектующей: ', CompatibilityNode^.Data.ComponentCode1);

WriteLn('Код второй комплектующей: ', CompatibilityNode^.Data.ComponentCode2);

WriteLn;

WriteLn('Введите новые данные:');

WriteLn('Доступные комплектующие:');

DisplayComponents(ComponentsList, ComponentTypesList);

repeat

ComponentCode1 := SafeReadInteger('Код первой комплектующей: ', 1);

if ComponentCode1 < 1 then

begin

WriteLn('Редактирование записи о совместимости отменено.');

PressEnterToContinue;

Exit;

end;

Component1 := FindComponent(ComponentsList, ComponentCode1);

if Component1 = nil then

WriteLn('Ошибка: Комплектующая с кодом ', ComponentCode1, ' не найдена. Пожалуйста, введите существующий код.');

until Component1 <> nil;

repeat

ComponentCode2 := SafeReadInteger('Код второй комплектующей: ', 1);

if ComponentCode2 < 1 then

begin

WriteLn('Редактирование записи о совместимости отменено.');

PressEnterToContinue;

Exit;

end;

Component2 := FindComponent(ComponentsList, ComponentCode2);

if Component2 = nil then

WriteLn('Ошибка: Комплектующая с кодом ', ComponentCode2, ' не найдена. Пожалуйста, введите существующий код.');

until Component2 <> nil;

if ComponentCode1 = ComponentCode2 then

begin

WriteLn('Ошибка: Нельзя создать запись о совместимости комплектующей с самой собой.');

PressEnterToContinue;

Exit;

end;

CompatibilityNode^.Data.ComponentCode1 := ComponentCode1;

CompatibilityNode^.Data.ComponentCode2 := ComponentCode2;

WriteLn('Запись о совместимости успешно отредактирована.');

end;

end;

PressEnterToContinue;

end;

// Процедура для обработки пункта меню "Специальные функции"

procedure HandleSpecialFunctions(const ComponentTypesList: TComponentTypeList;

const ComponentsList: TComponentList;

const CompatibilityList: TCompatibilityList;

var PCBuildOptionsList: TPCBuildOptionList;

var OrdersList: TOrderList);

var

SubmenuChoice: Integer;

MinPrice, MaxPrice: Real;

CompatibleComponentsList: TComponentList;

ComponentCode, TypeCode: Integer;

i: Integer;

BuildOptionNode: PPCBuildOptionNode;

begin

DisplaySpecialFunctionsSubmenu;

SubmenuChoice := GetSpecialFunctionsSubmenuChoice;

// Проверка на отмену операции

if SubmenuChoice < SUBMENU\_SF\_BUILD\_PC then

begin

WriteLn('Операция отменена.');

PressEnterToContinue;

Exit;

end;

WriteLn('Для прерывания операции в любой момент введите "/cancel" или "/menu"');

WriteLn;

case SubmenuChoice of

SUBMENU\_SF\_BUILD\_PC:

begin

WriteLn('=== ПОДБОР ВАРИАНТОВ КОМПЛЕКТАЦИИ ПК ===');

// Проверяем, есть ли необходимые данные

if ComponentsList.Count = 0 then

begin

WriteLn('Ошибка: Список комплектующих пуст. Сначала добавьте комплектующие.');

PressEnterToContinue;

Exit;

end;

if CompatibilityList.Count = 0 then

begin

WriteLn('Ошибка: Список совместимости пуст. Сначала добавьте записи о совместимости.');

PressEnterToContinue;

Exit;

end;

// Подбор вариантов комплектации ПК в заданном ценовом диапазоне

MinPrice := SafeReadFloat('Введите минимальную цену: ', 0);

if MinPrice < 0 then

begin

WriteLn('Подбор вариантов комплектации отменен.');

PressEnterToContinue;

Exit;

end;

MaxPrice := SafeReadFloat('Введите максимальную цену: ', MinPrice);

if MaxPrice < 0 then

begin

WriteLn('Подбор вариантов комплектации отменен.');

PressEnterToContinue;

Exit;

end;

if MaxPrice < MinPrice then

begin

WriteLn('Ошибка: Максимальная цена не может быть меньше минимальной.');

PressEnterToContinue;

Exit;

end;

if FindPCBuildOptions(ComponentsList, CompatibilityList, ComponentTypesList,

MinPrice, MaxPrice, PCBuildOptionsList) then

begin

WriteLn('Найдено ', PCBuildOptionsList.Count, ' вариантов комплектации ПК.');

DisplayPCBuildOptions(PCBuildOptionsList, ComponentsList);

// Сохраняем результаты в текстовый файл

if SavePCBuildOptionsToTextFile(PCBuildOptionsList, ComponentsList, 'data/pc\_build\_options.txt') then

WriteLn('Результаты сохранены в файл data/pc\_build\_options.txt')

else

WriteLn('Ошибка при сохранении результатов в файл.');

end

else

WriteLn('Не найдено вариантов комплектации ПК в заданном ценовом диапазоне.');

end;

SUBMENU\_SF\_CREATE\_ORDER:

begin

WriteLn('=== ОФОРМЛЕНИЕ ЗАКАЗА ===');

// Оформление заказа

if PCBuildOptionsList.Count = 0 then

begin

WriteLn('Сначала необходимо подобрать варианты комплектации ПК (пункт 1).');

PressEnterToContinue;

Exit;

end;

WriteLn('Доступные варианты комплектации:');

DisplayPCBuildOptions(PCBuildOptionsList, ComponentsList);

i := SafeReadInteger('Введите ID варианта комплектации для заказа: ', 1);

if i < 1 then

begin

WriteLn('Оформление заказа отменено.');

PressEnterToContinue;

Exit;

end;

BuildOptionNode := FindPCBuildOption(PCBuildOptionsList, i);

if BuildOptionNode = nil then

begin

WriteLn('Ошибка: Вариант комплектации с ID ', i, ' не найден.');

PressEnterToContinue;

Exit;

end;

if CreateOrder(BuildOptionNode^.Data, ComponentsList, OrdersList) then

WriteLn('Заказ успешно оформлен.')

else

WriteLn('Ошибка при оформлении заказа.');

end;

SUBMENU\_SF\_COMPATIBLE\_COMPONENTS:

begin

WriteLn('=== ПОИСК СОВМЕСТИМЫХ КОМПЛЕКТУЮЩИХ ===');

// Проверяем, есть ли необходимые данные

if ComponentsList.Count = 0 then

begin

WriteLn('Ошибка: Список комплектующих пуст.');

PressEnterToContinue;

Exit;

end;

if CompatibilityList.Count = 0 then

begin

WriteLn('Ошибка: Список совместимости пуст.');

PressEnterToContinue;

Exit;

end;

if ComponentTypesList.Count = 0 then

begin

WriteLn('Ошибка: Список типов комплектующих пуст.');

PressEnterToContinue;

Exit;

end;

// Поиск совместимых комплектующих заданного типа

WriteLn('Доступные комплектующие:');

DisplayComponents(ComponentsList, ComponentTypesList);

ComponentCode := SafeReadInteger('Введите код комплектующей: ', 1);

if ComponentCode < 1 then

begin

WriteLn('Поиск совместимых комплектующих отменен.');

PressEnterToContinue;

Exit;

end;

if FindComponent(ComponentsList, ComponentCode) = nil then

begin

WriteLn('Ошибка: Комплектующая с кодом ', ComponentCode, ' не найдена.');

PressEnterToContinue;

Exit;

end;

WriteLn('Доступные типы комплектующих:');

DisplayComponentTypes(ComponentTypesList);

TypeCode := SafeReadInteger('Введите код типа комплектующих для поиска совместимых: ', 1);

if TypeCode < 1 then

begin

WriteLn('Поиск совместимых комплектующих отменен.');

PressEnterToContinue;

Exit;

end;

if FindComponentType(ComponentTypesList, TypeCode) = nil then

begin

WriteLn('Ошибка: Тип комплектующих с кодом ', TypeCode, ' не найден.');

PressEnterToContinue;

Exit;

end;

if FindCompatibleComponents(ComponentsList, CompatibilityList, ComponentTypesList,

ComponentCode, TypeCode, CompatibleComponentsList) then

begin

WriteLn('Найдено ', CompatibleComponentsList.Count, ' совместимых комплектующих:');

DisplayComponents(CompatibleComponentsList, ComponentTypesList);

end

else

WriteLn('Не найдено совместимых комплектующих заданного типа.');

end;

end;

PressEnterToContinue;

end;

end.

======== SpecialFunctions.pas ========

unit SpecialFunctions;

{$mode objfpc}{$H+}

interface

uses

SysUtils, DataTypes, DynamicLists, FileOperations, UI;

{ Функция для подбора вариантов комплектации ПК в заданном ценовом диапазоне }

function FindPCBuildOptions(const ComponentList: TComponentList;

const CompatibilityList: TCompatibilityList;

const TypesList: TComponentTypeList;

MinPrice, MaxPrice: Real;

var BuildOptionsList: TPCBuildOptionList): Boolean;

{ Функция для оформления заказа }

function CreateOrder(const BuildOption: TPCBuildOption;

const ComponentList: TComponentList;

var OrdersList: TOrderList): Boolean;

{ Функция для поиска совместимых комплектующих заданного типа }

function FindCompatibleComponents(const ComponentList: TComponentList;

const CompatibilityList: TCompatibilityList;

const TypesList: TComponentTypeList;

ComponentCode, TypeCode: Integer;

var CompatibleComponents: TComponentList): Boolean;

implementation

{ Вспомогательная функция для проверки совместимости двух комплектующих }

function AreComponentsCompatible(ComponentCode1, ComponentCode2: Integer;

const CompatibilityList: TCompatibilityList): Boolean;

var

Current: PCompatibilityNode;

begin

Result := False;

Current := CompatibilityList.Head;

while Current <> nil do

begin

// Проверяем совместимость в обоих направлениях

if ((Current^.Data.ComponentCode1 = ComponentCode1) and (Current^.Data.ComponentCode2 = ComponentCode2)) or

((Current^.Data.ComponentCode1 = ComponentCode2) and (Current^.Data.ComponentCode2 = ComponentCode1)) then

begin

Result := True;

end;

Current := Current^.Next;

end;

end;

{ Вспомогательная функция для проверки совместимости комплектующей со всеми в наборе }

function IsCompatibleWithAll(ComponentCode: Integer;

const ComponentCodes: array of Integer;

const CompatibilityList: TCompatibilityList): Boolean;

var

i: Integer;

begin

Result := True;

for i := 0 to Length(ComponentCodes) - 1 do

begin

if not AreComponentsCompatible(ComponentCode, ComponentCodes[i], CompatibilityList) then

begin

Result := False;

Exit;

end;

end;

end;

{ Вспомогательная функция для получения списка комплектующих по типу }

function GetComponentsByType(const ComponentList: TComponentList;

TypeCode: Integer;

var ResultList: TComponentList): Boolean;

var

Current: PComponentNode;

begin

Result := False;

InitComponentList(ResultList);

Current := ComponentList.Head;

while Current <> nil do

begin

if Current^.Data.TypeCode = TypeCode then

AddComponent(ResultList, Current^.Data);

Current := Current^.Next;

end;

Result := ResultList.Count > 0;

end;

{ Вспомогательная функция для рекурсивного построения вариантов комплектации }

procedure BuildConfigurationsRecursive(const ComponentList: TComponentList;

const CompatibilityList: TCompatibilityList;

const TypesList: TComponentTypeList;

var BuildOptionsList: TPCBuildOptionList;

CurrentComponents: array of Integer;

CurrentPrice: Real;

MinPrice, MaxPrice: Real;

CurrentTypeIndex: Integer;

var NextBuildID: Integer);

var

TypeNode: PComponentTypeNode;

ComponentsOfType: TComponentList;

ComponentNode: PComponentNode;

NewComponents: array of Integer;

i, j: Integer;

BuildOption: TPCBuildOption;

begin

// Получаем текущий тип комплектующих

TypeNode := TypesList.Head;

for i := 1 to CurrentTypeIndex - 1 do

begin

if TypeNode = nil then

Exit;

TypeNode := TypeNode^.Next;

end;

// Если достигли конца списка типов, создаем вариант комплектации

if TypeNode = nil then

begin

// Проверяем, что цена в заданном диапазоне

if (CurrentPrice >= MinPrice) and (CurrentPrice <= MaxPrice) then

begin

// Создаем новый вариант комплектации

BuildOption.ID := NextBuildID;

Inc(NextBuildID);

SetLength(BuildOption.ComponentCodes, Length(CurrentComponents));

for i := 0 to Length(CurrentComponents) - 1 do

BuildOption.ComponentCodes[i] := CurrentComponents[i];

BuildOption.TotalPrice := CurrentPrice;

// Добавляем в список вариантов

AddPCBuildOption(BuildOptionsList, BuildOption);

end;

Exit;

end;

// Получаем все комплектующие текущего типа

if not GetComponentsByType(ComponentList, TypeNode^.Data.TypeCode, ComponentsOfType) then

begin

// Если нет комплектующих этого типа, переходим к следующему типу

BuildConfigurationsRecursive(ComponentList, CompatibilityList, TypesList,

BuildOptionsList, CurrentComponents, CurrentPrice,

MinPrice, MaxPrice, CurrentTypeIndex + 1, NextBuildID);

Exit;

end;

// Для каждой комплектующей текущего типа

ComponentNode := ComponentsOfType.Head;

while ComponentNode <> nil do

begin

// Проверяем совместимость с уже выбранными комплектующими

if IsCompatibleWithAll(ComponentNode^.Data.Code, CurrentComponents, CompatibilityList) then

begin

// Добавляем комплектующую к текущему набору

SetLength(NewComponents, Length(CurrentComponents) + 1);

for i := 0 to Length(CurrentComponents) - 1 do

NewComponents[i] := CurrentComponents[i];

NewComponents[Length(NewComponents) - 1] := ComponentNode^.Data.Code;

// Рекурсивно продолжаем построение для следующего типа

BuildConfigurationsRecursive(ComponentList, CompatibilityList, TypesList,

BuildOptionsList, NewComponents,

CurrentPrice + ComponentNode^.Data.Price,

MinPrice, MaxPrice, CurrentTypeIndex + 1, NextBuildID);

end;

ComponentNode := ComponentNode^.Next;

end;

// Освобождаем память

ClearComponentList(ComponentsOfType);

end;

{ Вспомогательная функция для сортировки вариантов комплектации по цене }

procedure SortBuildOptionsByPrice(var List: TPCBuildOptionList);

var

Sorted: Boolean;

Current, Next: PPCBuildOptionNode;

TempData: TPCBuildOption;

begin

if (List.Head = nil) or (List.Head^.Next = nil) then

Exit; // Список пуст или содержит только один элемент

repeat

Sorted := True;

Current := List.Head;

while (Current <> nil) and (Current^.Next <> nil) do

begin

Next := Current^.Next;

if Current^.Data.TotalPrice > Next^.Data.TotalPrice then

begin

// Меняем местами данные узлов

TempData := Current^.Data;

Current^.Data := Next^.Data;

Next^.Data := TempData;

Sorted := False;

end;

Current := Current^.Next;

end;

until Sorted;

end;

{ Реализация функции для подбора вариантов комплектации ПК в заданном ценовом диапазоне }

function FindPCBuildOptions(const ComponentList: TComponentList;

const CompatibilityList: TCompatibilityList;

const TypesList: TComponentTypeList;

MinPrice, MaxPrice: Real;

var BuildOptionsList: TPCBuildOptionList): Boolean;

var

EmptyArray: array of Integer;

NextBuildID: Integer;

begin

Result := False;

// Инициализируем список вариантов комплектации

InitPCBuildOptionList(BuildOptionsList);

// Инициализируем пустой массив комплектующих

SetLength(EmptyArray, 0);

// Получаем следующий доступный ID

NextBuildID := UI.GetNextID(BuildOptionsList);

// Рекурсивно строим все возможные варианты комплектации

BuildConfigurationsRecursive(ComponentList, CompatibilityList, TypesList,

BuildOptionsList, EmptyArray, 0,

MinPrice, MaxPrice, 1, NextBuildID);

// Сортируем варианты по цене

SortBuildOptionsByPrice(BuildOptionsList);

Result := BuildOptionsList.Count > 0;

end;

{ Реализация функции для оформления заказа }

function CreateOrder(const BuildOption: TPCBuildOption;

const ComponentList: TComponentList;

var OrdersList: TOrderList): Boolean;

var

Order: TOrder;

i: Integer;

ComponentNode: PComponentNode;

TempName, TempPhone: string;

begin

Result := False;

// Запрашиваем данные заказчика

Write('Введите имя заказчика: ');

ReadLn(TempName);

Order.CustomerName := StringToFixed(TempName);

Write('Введите телефон заказчика: ');

ReadLn(TempPhone);

Order.CustomerPhone := StringToFixed(TempPhone);

// Генерируем ID заказа с использованием функции GetNextID

Order.ID := UI.GetNextID(OrdersList);

// Устанавливаем ID варианта комплектации и дату заказа

Order.BuildOptionID := BuildOption.ID;

Order.OrderDate := Now;

// Проверяем наличие комплектующих и уменьшаем их количество

for i := 0 to Length(BuildOption.ComponentCodes) - 1 do

begin

ComponentNode := FindComponent(ComponentList, BuildOption.ComponentCodes[i]);

if ComponentNode = nil then

begin

WriteLn('Ошибка: Комплектующая с кодом ', BuildOption.ComponentCodes[i], ' не найдена.');

Exit;

end;

if ComponentNode^.Data.InStock <= 0 then

begin

WriteLn('Ошибка: Комплектующая ', ComponentNode^.Data.Manufacturer, ' ',

ComponentNode^.Data.Model, ' отсутствует на складе.');

Exit;

end;

// Уменьшаем количество на складе

Dec(ComponentNode^.Data.InStock);

end;

// Добавляем заказ в список

if not AddOrder(OrdersList, Order) then

begin

WriteLn('Ошибка при добавлении заказа в список.');

Exit;

end;

// Сохраняем информацию о заказе в текстовый файл

if not SaveOrderToTextFile(Order, BuildOption, ComponentList, '') then

begin

WriteLn('Ошибка при сохранении информации о заказе в файл.');

Exit;

end;

Result := True;

end;

{ Реализация функции для поиска совместимых комплектующих заданного типа }

function FindCompatibleComponents(const ComponentList: TComponentList;

const CompatibilityList: TCompatibilityList;

const TypesList: TComponentTypeList;

ComponentCode, TypeCode: Integer;

var CompatibleComponents: TComponentList): Boolean;

var

BaseComponent: PComponentNode;

Current: PComponentNode;

begin

Result := False;

// Инициализируем список совместимых комплектующих

InitComponentList(CompatibleComponents);

// Находим базовую комплектующую

BaseComponent := FindComponent(ComponentList, ComponentCode);

if BaseComponent = nil then

begin

WriteLn('Ошибка: Комплектующая с кодом ', ComponentCode, ' не найдена.');

Exit;

end;

// Проходим по всем комплектующим заданного типа

Current := ComponentList.Head;

while Current <> nil do

begin

// Проверяем, что комплектующая имеет заданный тип и совместима с базовой

if (Current^.Data.TypeCode = TypeCode) and

AreComponentsCompatible(BaseComponent^.Data.Code, Current^.Data.Code, CompatibilityList) then

begin

// Добавляем в список совместимых

AddComponent(CompatibleComponents, Current^.Data);

end;

Current := Current^.Next;

end;

Result := CompatibleComponents.Count > 0;

end;

end.

======== UI.pas ========

unit UI;

{$mode objfpc}{$H+}

interface

uses

SysUtils, Crt, DataTypes, DynamicLists, InputUtils;

{ Константы для пунктов меню }

const

MENU\_LOAD\_DATA = 1;

MENU\_VIEW\_LIST = 2;

MENU\_SORT\_DATA = 3;

MENU\_SEARCH\_DATA = 4;

MENU\_ADD\_DATA = 5;

MENU\_REMOVE\_DATA = 6;

MENU\_EDIT\_DATA = 7;

MENU\_SPECIAL\_FUNCTIONS = 8;

MENU\_EXIT\_WITHOUT\_SAVE = 9;

MENU\_EXIT\_WITH\_SAVE = 10;

{ Константы для подменю списков }

const

SUBMENU\_COMPONENT\_TYPES = 1;

SUBMENU\_COMPONENTS = 2;

SUBMENU\_COMPATIBILITY = 3;

SUBMENU\_ORDERS = 4;

{ Константы для подменю специальных функций }

const

SUBMENU\_SF\_BUILD\_PC = 1;

SUBMENU\_SF\_CREATE\_ORDER = 2;

SUBMENU\_SF\_COMPATIBLE\_COMPONENTS = 3;

{ Процедуры для отображения меню и подменю }

procedure DisplayMainMenu;

function GetMenuChoice: Integer;

procedure DisplayListSubmenu;

function GetListSubmenuChoice: Integer;

procedure DisplayAddDataSubmenu;

function GetAddDataSubmenuChoice: Integer;

procedure DisplaySpecialFunctionsSubmenu;

function GetSpecialFunctionsSubmenuChoice: Integer;

{ Процедуры для отображения данных }

procedure DisplayComponentTypes(const List: TComponentTypeList);

procedure DisplayComponents(const List: TComponentList; const TypesList: TComponentTypeList);

procedure DisplayCompatibility(const List: TCompatibilityList; const ComponentsList: TComponentList);

procedure DisplayOrders(const List: TOrderList; const BuildOptionsList: TPCBuildOptionList);

procedure DisplayOrdersDetailed(const List: TOrderList; const BuildOptionsList: TPCBuildOptionList; const ComponentsList: TComponentList);

procedure DisplayPCBuildOptions(const List: TPCBuildOptionList; const ComponentsList: TComponentList);

{ Процедуры для ввода данных }

function InputComponentType(var ComponentType: TComponentType; const List: TComponentTypeList): Boolean;

function InputComponent(var Component: TComponent; const TypesList: TComponentTypeList; const List: TComponentList): Boolean;

function InputCompatibility(var Compatibility: TCompatibility; const ComponentsList: TComponentList): Boolean;

function InputOrder(var Order: TOrder; const BuildOptionsList: TPCBuildOptionList; const List: TOrderList): Boolean;

{ Вспомогательные процедуры }

procedure ClearScreen;

procedure PressEnterToContinue;

function GetNextID(const List: TComponentTypeList): Integer; overload;

function GetNextID(const List: TComponentList): Integer; overload;

function GetNextID(const List: TPCBuildOptionList): Integer; overload;

function GetNextID(const List: TOrderList): Integer; overload;

function GetComponentTypeName(TypeCode: Integer; const TypesList: TComponentTypeList): string;

{ Глобальные счетчики для уникальных ID }

var

NextComponentTypeID: Integer = 1;

NextComponentID: Integer = 1;

NextPCBuildOptionID: Integer = 1;

NextOrderID: Integer = 1;

implementation

{ Реализация процедур для отображения меню и подменю }

procedure DisplayMainMenu;

begin

ClearScreen;

WriteLn('=== ГЛАВНОЕ МЕНЮ ===');

WriteLn('1. Чтение данных из файла');

WriteLn('2. Просмотр всего списка');

WriteLn('3. Сортировка данных');

WriteLn('4. Поиск данных с использованием фильтров');

WriteLn('5. Добавление данных в список');

WriteLn('6. Удаление данных из списка');

WriteLn('7. Редактирование данных');

WriteLn('8. Сборка ПК и Заказы');

WriteLn('9. Выход из программы без сохранения изменений');

WriteLn('10. Выход с сохранением изменений');

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

Write('Выберите пункт меню: ');

end;

function GetMenuChoice: Integer;

begin

Result := SafeReadInteger('', MENU\_LOAD\_DATA, MENU\_EXIT\_WITH\_SAVE,

'Ошибка: Выберите пункт меню от 1 до 10.');

end;

procedure DisplayListSubmenu;

begin

ClearScreen;

WriteLn('=== ВЫБЕРИТЕ СПИСОК ===');

WriteLn('1. Типы комплектующих');

WriteLn('2. Комплектующие');

WriteLn('3. Совместимость комплектующих');

WriteLn('4. Заказы');

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

Write('Выберите список: ');

end;

function GetListSubmenuChoice: Integer;

begin

Result := SafeReadInteger('', SUBMENU\_COMPONENT\_TYPES, SUBMENU\_ORDERS,

'Ошибка: Выберите пункт подменю от 1 до 4.');

end;

procedure DisplayAddDataSubmenu;

begin

ClearScreen;

WriteLn('=== ВЫБЕРИТЕ ТИП ДАННЫХ ДЛЯ ДОБАВЛЕНИЯ ===');

WriteLn('1. Типы комплектующих');

WriteLn('2. Комплектующие');

WriteLn('3. Совместимость комплектующих');

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

Write('Выберите тип данных: ');

end;

function GetAddDataSubmenuChoice: Integer;

begin

Result := SafeReadInteger('', SUBMENU\_COMPONENT\_TYPES, SUBMENU\_COMPATIBILITY,

'Ошибка: Выберите пункт подменю от 1 до 3.');

end;

procedure DisplaySpecialFunctionsSubmenu;

begin

ClearScreen;

WriteLn('=== СПЕЦИАЛЬНЫЕ ФУНКЦИИ ===');

WriteLn('1. Подбор вариантов комплектации ПК в заданном ценовом диапазоне');

WriteLn('2. Оформление заказа');

WriteLn('3. Поиск совместимых комплектующих заданного типа');

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

Write('Выберите функцию: ');

end;

function GetSpecialFunctionsSubmenuChoice: Integer;

begin

Result := SafeReadInteger('', SUBMENU\_SF\_BUILD\_PC, SUBMENU\_SF\_COMPATIBLE\_COMPONENTS,

'Ошибка: Выберите пункт подменю от 1 до 3.');

end;

{ Реализация процедур для отображения данных }

procedure DisplayComponentTypes(const List: TComponentTypeList);

var

Current: PComponentTypeNode;

Count: Integer;

begin

ClearScreen;

WriteLn('=== СПИСОК ТИПОВ КОМПЛЕКТУЮЩИХ ===');

WriteLn('Код типа │ Название');

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

Current := List.Head;

Count := 0;

while Current <> nil do

begin

WriteLn(Current^.Data.TypeCode:8, ' │ ', Current^.Data.Name);

Current := Current^.Next;

Inc(Count);

end;

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

WriteLn('Всего записей: ', Count);

WriteLn;

PressEnterToContinue;

end;

procedure DisplayComponents(const List: TComponentList; const TypesList: TComponentTypeList);

var

Current: PComponentNode;

Count: Integer;

TypeName: string;

begin

ClearScreen;

WriteLn('=== СПИСОК КОМПЛЕКТУЮЩИХ ===');

WriteLn('Код │ Тип │ Производитель │ Модель │ Параметры │ Цена │ В наличии');

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

Current := List.Head;

Count := 0;

while Current <> nil do

begin

TypeName := GetComponentTypeName(Current^.Data.TypeCode, TypesList);

WriteLn(Current^.Data.Code:3, ' │ ',

TypeName:10, ' │ ',

Current^.Data.Manufacturer:13, ' │ ',

Current^.Data.Model:10, ' │ ',

Current^.Data.Parameters:15, ' │ ',

Current^.Data.Price:8:2, ' │ ',

Current^.Data.InStock:9);

Current := Current^.Next;

Inc(Count);

end;

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

WriteLn('Всего записей: ', Count);

WriteLn;

PressEnterToContinue;

end;

procedure DisplayCompatibility(const List: TCompatibilityList; const ComponentsList: TComponentList);

var

Current: PCompatibilityNode;

Count: Integer;

Component1, Component2: PComponentNode;

begin

ClearScreen;

WriteLn('=== СПИСОК СОВМЕСТИМОСТИ КОМПЛЕКТУЮЩИХ ===');

WriteLn('Код 1 │ Комплектующая 1 │ Код 2 │ Комплектующая 2');

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

Current := List.Head;

Count := 0;

while Current <> nil do

begin

Component1 := FindComponent(ComponentsList, Current^.Data.ComponentCode1);

Component2 := FindComponent(ComponentsList, Current^.Data.ComponentCode2);

Write(Current^.Data.ComponentCode1:5, ' │ ');

if Component1 <> nil then

Write((Component1^.Data.Manufacturer + ' ' + Component1^.Data.Model):20)

else

Write('Неизвестно':20);

Write(' │ ', Current^.Data.ComponentCode2:5, ' │ ');

if Component2 <> nil then

WriteLn((Component2^.Data.Manufacturer + ' ' + Component2^.Data.Model):20)

else

WriteLn('Неизвестно':20);

Current := Current^.Next;

Inc(Count);

end;

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

WriteLn('Всего записей: ', Count);

WriteLn;

PressEnterToContinue;

end;

procedure DisplayOrders(const List: TOrderList; const BuildOptionsList: TPCBuildOptionList);

var

Current: POrderNode;

Count: Integer;

BuildOption: PPCBuildOptionNode;

begin

ClearScreen;

WriteLn('=== СПИСОК ЗАКАЗОВ ===');

WriteLn('ID │ Вариант │ Заказчик │ Телефон │ Дата заказа │ Стоимость │ Компонентов');

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

Current := List.Head;

Count := 0;

while Current <> nil do

begin

BuildOption := FindPCBuildOption(BuildOptionsList, Current^.Data.BuildOptionID);

Write(Current^.Data.ID:2, ' │ ',

Current^.Data.BuildOptionID:7, ' │ ',

Current^.Data.CustomerName:12, ' │ ',

Current^.Data.CustomerPhone:8, ' │ ',

FormatDateTime('dd.mm.yyyy', Current^.Data.OrderDate):11, ' │ ');

if BuildOption <> nil then

begin

Write(BuildOption^.Data.TotalPrice:9:2, ' │ ');

WriteLn(Length(BuildOption^.Data.ComponentCodes):11);

end

else

WriteLn(' Неизвестно │ Неизвестно');

Current := Current^.Next;

Inc(Count);

end;

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

WriteLn('Всего записей: ', Count);

WriteLn;

PressEnterToContinue;

end;

procedure DisplayOrdersDetailed(const List: TOrderList; const BuildOptionsList: TPCBuildOptionList; const ComponentsList: TComponentList);

var

Current: POrderNode;

Count, i: Integer;

BuildOption: PPCBuildOptionNode;

ComponentNode: PComponentNode;

begin

ClearScreen;

WriteLn('=== ПОДРОБНЫЙ СПИСОК ЗАКАЗОВ ===');

Current := List.Head;

Count := 0;

while Current <> nil do

begin

Inc(Count);

WriteLn;

WriteLn('Заказ #', Current^.Data.ID);

WriteLn('══════════════════════════════════════════════════════════════');

WriteLn('Заказчик: ', Current^.Data.CustomerName);

WriteLn('Телефон: ', Current^.Data.CustomerPhone);

WriteLn('Дата заказа: ', FormatDateTime('dd.mm.yyyy hh:nn', Current^.Data.OrderDate));

WriteLn('Вариант сборки: #', Current^.Data.BuildOptionID);

BuildOption := FindPCBuildOption(BuildOptionsList, Current^.Data.BuildOptionID);

if BuildOption <> nil then

begin

WriteLn('Общая стоимость: ', BuildOption^.Data.TotalPrice:0:2, ' руб.');

WriteLn('Количество компонентов: ', Length(BuildOption^.Data.ComponentCodes));

WriteLn('Состав комплектации:');

for i := 0 to Length(BuildOption^.Data.ComponentCodes) - 1 do

begin

ComponentNode := FindComponent(ComponentsList, BuildOption^.Data.ComponentCodes[i]);

if ComponentNode <> nil then

begin

WriteLn(' • ', ComponentNode^.Data.Manufacturer, ' ',

ComponentNode^.Data.Model, ' — ',

ComponentNode^.Data.Price:0:2, ' руб.');

end

else

WriteLn(' • Неизвестный компонент (код: ', BuildOption^.Data.ComponentCodes[i], ')');

end;

end

else

begin

WriteLn('Общая стоимость: Неизвестно (вариант сборки не найден)');

WriteLn('Состав комплектации: Недоступен');

end;

Current := Current^.Next;

end;

WriteLn;

WriteLn('══════════════════════════════════════════════════════════════');

WriteLn('Всего заказов: ', Count);

WriteLn;

PressEnterToContinue;

end;

procedure DisplayPCBuildOptions(const List: TPCBuildOptionList; const ComponentsList: TComponentList);

var

Current: PPCBuildOptionNode;

Count, i: Integer;

ComponentNode: PComponentNode;

begin

ClearScreen;

WriteLn('=== ВАРИАНТЫ КОМПЛЕКТАЦИИ ПК ===');

Current := List.Head;

Count := 0;

while Current <> nil do

begin

WriteLn('Вариант #', Current^.Data.ID);

WriteLn('Общая стоимость: ', Current^.Data.TotalPrice:0:2, ' руб.');

WriteLn('Комплектующие:');

for i := 0 to Length(Current^.Data.ComponentCodes) - 1 do

begin

ComponentNode := FindComponent(ComponentsList, Current^.Data.ComponentCodes[i]);

if ComponentNode <> nil then

begin

WriteLn(' • ', ComponentNode^.Data.Manufacturer, ' ',

ComponentNode^.Data.Model, ' (',

ComponentNode^.Data.Price:0:2, ' руб.)');

end;

end;

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

Current := Current^.Next;

Inc(Count);

end;

WriteLn('Всего вариантов: ', Count);

WriteLn;

PressEnterToContinue;

end;

{ Реализация процедур для ввода данных }

function InputComponentType(var ComponentType: TComponentType; const List: TComponentTypeList): Boolean;

var

TempName: string;

begin

Result := False;

// Автоматически генерируем код типа комплектующей

ComponentType.TypeCode := GetNextID(List);

WriteLn('Код типа комплектующей (генерируется автоматически): ', ComponentType.TypeCode);

// Безопасный ввод названия типа комплектующей

TempName := SafeReadString('Введите название типа комплектующей: ', 1, MAX\_STRING\_LENGTH);

// Проверка на команду возврата в меню

if IsReturnToMenuCommand(TempName) or IsCancelCommand(TempName) then

Exit;

ComponentType.Name := StringToFixed(TempName);

Result := True;

end;

function InputComponent(var Component: TComponent; const TypesList: TComponentTypeList; const List: TComponentList): Boolean;

var

TypeNode: PComponentTypeNode;

TempManufacturer, TempModel, TempParams: string;

TypeCode: Integer;

begin

Result := False;

// Автоматически генерируем код комплектующей

Component.Code := GetNextID(List);

WriteLn('Код комплектующей (генерируется автоматически): ', Component.Code);

// Ввод и проверка кода типа

repeat

TypeCode := SafeReadInteger('Введите код типа комплектующей: ', 1);

// Проверка на команду возврата в меню

if TypeCode < 1 then

Exit;

Component.TypeCode := TypeCode;

TypeNode := FindComponentType(TypesList, Component.TypeCode);

if TypeNode = nil then

WriteLn('Ошибка: Тип с кодом ', Component.TypeCode, ' не найден. Пожалуйста, введите существующий код типа.');

until TypeNode <> nil;

// Безопасный ввод фирмы-изготовителя

TempManufacturer := SafeReadString('Введите фирму-изготовителя: ', 1, MAX\_STRING\_LENGTH);

if IsReturnToMenuCommand(TempManufacturer) or IsCancelCommand(TempManufacturer) then

Exit;

Component.Manufacturer := StringToFixed(TempManufacturer);

// Безопасный ввод модели

TempModel := SafeReadString('Введите модель: ', 1, MAX\_STRING\_LENGTH);

if IsReturnToMenuCommand(TempModel) or IsCancelCommand(TempModel) then

Exit;

Component.Model := StringToFixed(TempModel);

// Безопасный ввод параметров

TempParams := SafeReadString('Введите параметры: ', 0, MAX\_STRING\_LENGTH);

if IsReturnToMenuCommand(TempParams) or IsCancelCommand(TempParams) then

Exit;

Component.Parameters := StringToFixed(TempParams);

// Безопасный ввод цены

Component.Price := SafeReadFloat('Введите цену: ', 0);

if Component.Price < 0 then

Exit;

// Безопасный ввод количества в наличии

Component.InStock := SafeReadInteger('Введите количество в наличии: ', 0);

if Component.InStock < 0 then

Exit;

Result := True;

end;

function InputCompatibility(var Compatibility: TCompatibility; const ComponentsList: TComponentList): Boolean;

var

Component1, Component2: PComponentNode;

ComponentCode1, ComponentCode2: Integer;

begin

Result := False;

try

// Ввод и проверка кода первой комплектующей

repeat

ComponentCode1 := SafeReadInteger('Введите код первой комплектующей: ', 1);

// Проверка на команду возврата в меню или отмены

if ComponentCode1 < 1 then

Exit;

Compatibility.ComponentCode1 := ComponentCode1;

Component1 := FindComponent(ComponentsList, Compatibility.ComponentCode1);

if Component1 = nil then

WriteLn('Ошибка: Комплектующая с кодом ', Compatibility.ComponentCode1, ' не найдена. Пожалуйста, введите существующий код.');

until Component1 <> nil;

// Ввод и проверка кода второй комплектующей

repeat

ComponentCode2 := SafeReadInteger('Введите код второй комплектующей (совместимой с первой): ', 1);

// Проверка на команду возврата в меню или отмены

if ComponentCode2 < 1 then

Exit;

Compatibility.ComponentCode2 := ComponentCode2;

Component2 := FindComponent(ComponentsList, Compatibility.ComponentCode2);

if Component2 = nil then

WriteLn('Ошибка: Комплектующая с кодом ', Compatibility.ComponentCode2, ' не найдена. Пожалуйста, введите существующий код.');

until Component2 <> nil;

// Проверка, что коды различны

if Compatibility.ComponentCode1 = Compatibility.ComponentCode2 then

begin

WriteLn('Ошибка: Нельзя создать запись о совместимости комплектующей с самой собой.');

Exit;

end;

Result := True;

except

WriteLn('Ошибка при вводе данных.');

end;

end;

function InputOrder(var Order: TOrder; const BuildOptionsList: TPCBuildOptionList; const List: TOrderList): Boolean;

var

BuildOption: PPCBuildOptionNode;

TempName, TempPhone: string;

BuildOptionID: Integer;

begin

Result := False;

try

// Автоматически генерируем ID заказа

Order.ID := GetNextID(List);

WriteLn('ID заказа (генерируется автоматически): ', Order.ID);

// Ввод и проверка ID варианта комплектации

repeat

BuildOptionID := SafeReadInteger('Введите ID варианта комплектации: ', 1);

// Проверка на команду возврата в меню или отмены

if BuildOptionID < 1 then

Exit;

Order.BuildOptionID := BuildOptionID;

BuildOption := FindPCBuildOption(BuildOptionsList, Order.BuildOptionID);

if BuildOption = nil then

WriteLn('Ошибка: Вариант комплектации с ID ', Order.BuildOptionID, ' не найден. Пожалуйста, введите существующий ID.');

until BuildOption <> nil;

TempName := SafeReadString('Введите имя заказчика: ', 1, MAX\_STRING\_LENGTH);

if IsReturnToMenuCommand(TempName) or IsCancelCommand(TempName) then

Exit;

Order.CustomerName := StringToFixed(TempName);

TempPhone := SafeReadString('Введите телефон заказчика: ', 1, MAX\_STRING\_LENGTH);

if IsReturnToMenuCommand(TempPhone) or IsCancelCommand(TempPhone) then

Exit;

Order.CustomerPhone := StringToFixed(TempPhone);

// Устанавливаем текущую дату и время

Order.OrderDate := Now;

Result := True;

except

WriteLn('Ошибка при вводе данных.');

end;

end;

{ Реализация вспомогательных процедур }

procedure ClearScreen;

begin

ClrScr;

end;

procedure PressEnterToContinue;

begin

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

ReadLn;

end;

function GetNextID(const List: TComponentTypeList): Integer;

var

Current: PComponentTypeNode;

MaxID: Integer;

begin

// Инициализируем глобальный счетчик на основе существующих данных (только при первом вызове)

if NextComponentTypeID = 1 then

begin

MaxID := 0;

Current := List.Head;

while Current <> nil do

begin

if Current^.Data.TypeCode > MaxID then

MaxID := Current^.Data.TypeCode;

Current := Current^.Next;

end;

NextComponentTypeID := MaxID + 1;

end;

Result := NextComponentTypeID;

Inc(NextComponentTypeID);

end;

function GetNextID(const List: TComponentList): Integer;

var

Current: PComponentNode;

MaxID: Integer;

begin

// Инициализируем глобальный счетчик на основе существующих данных (только при первом вызове)

if NextComponentID = 1 then

begin

MaxID := 0;

Current := List.Head;

while Current <> nil do

begin

if Current^.Data.Code > MaxID then

MaxID := Current^.Data.Code;

Current := Current^.Next;

end;

NextComponentID := MaxID + 1;

end;

Result := NextComponentID;

Inc(NextComponentID);

end;

function GetNextID(const List: TPCBuildOptionList): Integer;

var

Current: PPCBuildOptionNode;

MaxID: Integer;

begin

// Инициализируем глобальный счетчик на основе существующих данных (только при первом вызове)

if NextPCBuildOptionID = 1 then

begin

MaxID := 0;

Current := List.Head;

while Current <> nil do

begin

if Current^.Data.ID > MaxID then

MaxID := Current^.Data.ID;

Current := Current^.Next;

end;

NextPCBuildOptionID := MaxID + 1;

end;

Result := NextPCBuildOptionID;

Inc(NextPCBuildOptionID);

end;

function GetNextID(const List: TOrderList): Integer;

var

Current: POrderNode;

MaxID: Integer;

begin

// Инициализируем глобальный счетчик на основе существующих данных (только при первом вызове)

if NextOrderID = 1 then

begin

MaxID := 0;

Current := List.Head;

while Current <> nil do

begin

if Current^.Data.ID > MaxID then

MaxID := Current^.Data.ID;

Current := Current^.Next;

end;

NextOrderID := MaxID + 1;

end;

Result := NextOrderID;

Inc(NextOrderID);

end;

function GetComponentTypeName(TypeCode: Integer; const TypesList: TComponentTypeList): string;

var

TypeNode: PComponentTypeNode;

begin

TypeNode := FindComponentType(TypesList, TypeCode);

if TypeNode <> nil then

Result := TypeNode^.Data.Name

else

Result := 'Неизвестно';

end;

end.

======== solve.dpr ========

program solve;

{$mode objfpc}{$H+}

uses

SysUtils,

DataTypes,

DynamicLists,

FileOperations,

UI,

SpecialFunctions,

MenuHandlers,

MainProgram;

var

// Списки данных

ComponentTypesList: TComponentTypeList;

ComponentsList: TComponentList;

CompatibilityList: TCompatibilityList;

PCBuildOptionsList: TPCBuildOptionList;

OrdersList: TOrderList;

// Переменные для работы с меню

MenuChoice: Integer;

ExitProgram: Boolean;

begin

// Инициализация

InitAllLists(ComponentTypesList, ComponentsList, CompatibilityList,

PCBuildOptionsList, OrdersList);

ExitProgram := False;

// Главный цикл программы

repeat

DisplayMainMenu;

MenuChoice := GetMenuChoice;

case MenuChoice of

MENU\_LOAD\_DATA:

LoadAllData(ComponentTypesList, ComponentsList, CompatibilityList, OrdersList);

MENU\_VIEW\_LIST:

ViewList(ComponentTypesList, ComponentsList, CompatibilityList,

OrdersList, PCBuildOptionsList);

MENU\_SORT\_DATA:

HandleSortData(ComponentTypesList, ComponentsList, OrdersList);

MENU\_SEARCH\_DATA:

HandleSearchData(ComponentsList, ComponentTypesList);

MENU\_ADD\_DATA:

AddData(ComponentTypesList, ComponentsList, CompatibilityList);

MENU\_REMOVE\_DATA:

RemoveData(ComponentTypesList, ComponentsList, CompatibilityList);

MENU\_EDIT\_DATA:

EditData(ComponentTypesList, ComponentsList, CompatibilityList);

MENU\_SPECIAL\_FUNCTIONS:

HandleSpecialFunctions(ComponentTypesList, ComponentsList, CompatibilityList,

PCBuildOptionsList, OrdersList);

MENU\_EXIT\_WITHOUT\_SAVE:

begin

WriteLn('Выход без сохранения изменений.');

ExitProgram := True;

end;

MENU\_EXIT\_WITH\_SAVE:

begin

SaveAllData(ComponentTypesList, ComponentsList, CompatibilityList, OrdersList);

WriteLn('Выход с сохранением изменений.');

ExitProgram := True;

end;

end;

until ExitProgram;

// Освобождение памяти

ClearComponentTypeList(ComponentTypesList);

ClearComponentList(ComponentsList);

ClearCompatibilityList(CompatibilityList);

ClearPCBuildOptionList(PCBuildOptionsList);

ClearOrderList(OrdersList);

end.