МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное агентство по образованию  
«Санкт-Петербургский государственный университет телекоммуникаций  
им. проф. М. А. Бонч-Бруевича (СПБГУТ)

Факультет информационных технологий и программной инженерии.

СПбГУТ **)))**

Курсовая работа по дисциплине:

Объектно-ориентированное программирование

Вариант 24

Выполнил:

Студент 2 курса, группа

ИКПИ-35

***Тетеркин Георгий Михайлович***

*Преподаватель:*

***Коробов Сергей Александрович***

Санкт-Петербург

**2024**

**Оглавление**

[**1.** **ПОСТАНОВКА ЗАДАЧИ** 3](#_Toc185489895)

[**2.** **ВЫБОР И ОБОСНОВАНИЕ ТИПА РАЗРАБАТЫВАЕМОГО КОНТЕЙНЕРА** 4](#_Toc185489896)

[Обоснование выбора std::vector: 4](#_Toc185489897)

[Альтернативы и их недостатки в данном случае: 4](#_Toc185489898)

[**3.** **РАЗРАБОТКА СТРУКТУРЫ ПРОГРАММЫ** 5](#_Toc185489899)

[Общая структура программы 5](#_Toc185489900)

[Архитектура программы 5](#_Toc185489901)

[Графическая структура программы 6](#_Toc185489902)

[**4.** **ВЫБОР И ОБОСНОВАНИЕ ИСПОЛЬЗУЕМЫХ КЛАССОВ** 8](#_Toc185489903)

[**4.1 MainWindow** 8](#_Toc185489904)

[**4.2 PhoneModel** 8](#_Toc185489905)

[**4.3 PhoneDatabase** 9](#_Toc185489906)

[**4.4 PhoneEditor** 9](#_Toc185489907)

[**4.5 Связь классов** 10](#_Toc185489908)

[**5.** **РАЗРАБОТКА СОБСТВЕННЫХ КЛАССОВ** 10](#_Toc185489909)

[1. PhoneModel 10](#_Toc185489910)

[2. PhoneDatabase 11](#_Toc185489911)

[3. PhoneManager 11](#_Toc185489912)

[4. PhoneEditorDialog 11](#_Toc185489913)

[**6.** **РАЗРАБОТКА ИНТЕРФЕЙСА ПРОГРАММЫ** 13](#_Toc185489914)

[Основное окно (MainWindow) 13](#_Toc185489915)

[Диалоговое окно (PhoneEditorDialog) 15](#_Toc185489916)

[Структура интерфейса 15](#_Toc185489917)

[**7.** **ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЮ** 16](#_Toc185489918)

[Запуск программы 16](#_Toc185489919)

[Элементы интерфейса 16](#_Toc185489920)

[Работа с программой 17](#_Toc185489921)

[Поиск записей 19](#_Toc185489922)

[Дополнительные функции 20](#_Toc185489923)

[**8.** **ЗАКЛЮЧЕНИЕ** 21](#_Toc185489924)

[Достоинства программы 21](#_Toc185489925)

[Перспективы развития 21](#_Toc185489926)

[Заключение 22](#_Toc185489927)

[**9.** **СПИСОК ЛИТЕРАТУРЫ** 23](#_Toc185489928)

[10.1 MainWindow.h 23](#_Toc185489929)

[10.2 MainWindow.cpp 24](#_Toc185489930)

[10.3 PhoneModel.h 27](#_Toc185489931)

[10.4 PhoneModel.cpp 28](#_Toc185489932)

[10.5 PhoneDatabase.h 28](#_Toc185489933)

[10.6 PhoneDatabase.cpp 29](#_Toc185489934)

[10.7 PhoneEditorDialog.h 30](#_Toc185489935)

[10.8 PhoneEditorDialog.cpp 31](#_Toc185489936)

[10.9 PhoneEditorDialog.ui 32](#_Toc185489937)

# **ПОСТАНОВКА ЗАДАЧИ**

Разработать приложение для управления базой данных смартфонов, используя язык программирования C++ и библиотеку Qt. Программа должна:

* Позволять добавлять, редактировать, удалять записи о смартфонах.
* Содержать поисковую функцию по различным критериям.
* Обеспечивать возможность сохранения базы данных в файл и загрузки из файла.
* Предоставлять удобный графический интерфейс, включающий главное меню, панель инструментов и окно "О программе".

Элементы базы данных — это записи о смартфонах, включающие следующие параметры:

* Модель смартфона.
* Производитель.
* Операционная система.
* Объём памяти (ГБ).
* Цена.

Программа должна обеспечивать:

1. Создание новой базы данных.
2. Добавление, редактирование и удаление записей.
3. Поиск записей по критериям.
4. Сохранение базы данных в файл.
5. Загрузку базы данных из файла.
6. Удобный пользовательский интерфейс для взаимодействия с программой.

# **ВЫБОР И ОБОСНОВАНИЕ ТИПА РАЗРАБАТЫВАЕМОГО КОНТЕЙНЕРА**

Для реализации базы данных смартфонов в данном приложении выбран контейнер на основе стандартной библиотеки C++ (STL) — std::vector. Это решение обусловлено следующими причинами:

### **Обоснование выбора** std::vector:

1. **Простота использования**:  
   Контейнер std::vector предоставляет удобный интерфейс для работы с динамическими массивами, что упрощает добавление, удаление и доступ к элементам.
2. **Производительность**:
   * Вектор обеспечивает быструю (O(1)) индексацию по элементам благодаря последовательному расположению данных в памяти.
   * Операции добавления элемента в конец контейнера выполняются в среднем за O(1).
3. **Динамическое изменение размера**:  
   В отличие от обычных массивов, std::vector автоматически изменяет свой размер при добавлении новых элементов, что важно для хранения записей о смартфонах в динамически изменяемой базе данных.
4. **Совместимость с итераторами**:  
   Контейнер поддерживает итераторы, которые используются для эффективного выполнения операций поиска, сортировки и других манипуляций с данными.
5. **Соответствие требованиям проекта**:
   * Приложение использует операции добавления, удаления и поиска данных. Для этого std::vectorобеспечивает достаточную производительность и простоту реализации.
   * При необходимости вектор легко заменяется на другой контейнер (например, std::list или std::deque) без значительных изменений в архитектуре приложения.

### **Альтернативы и их недостатки в данном случае:**

1. **std::list**:
   * Предоставляет быструю вставку и удаление элементов, но доступ по индексу выполняется за O(n), что снижает производительность при необходимости частого обращения к элементам.
   * Занимает больше памяти из-за дополнительных указателей.
2. **std::map или std::unordered\_map**:
   * Эти контейнеры подходят для структурированных данных, где ключи уникальны. Однако в данном случае необходимость в ключах отсутствует, так как база данных представляет собой линейную коллекцию записей.

# **РАЗРАБОТКА СТРУКТУРЫ ПРОГРАММЫ**

Программа для управления базой данных смартфонов выполнена с использованием принципов объектно-ориентированного программирования (ООП). Её модульная структура обеспечивает лёгкость поддержки и расширяемость.

### **Общая структура программы**

1. **Главное окно (MainWindow)**:
   * Центральный компонент программы, через который пользователь взаимодействует с базой данных.
   * Управляет таблицей с данными и предоставляет доступ ко всем функциям приложения.
2. **Класс модели данных (PhoneModel)**:
   * Описывает объект базы данных — смартфон.
3. **Контейнер для хранения данных (PhoneDatabase)**:
   * Хранит коллекцию объектов PhoneModel и предоставляет методы для добавления, редактирования, удаления и поиска данных.
4. **Диалоговое окно (PhoneEditor)**:
   * Обеспечивает интерфейс для ввода данных о смартфоне.

### **Архитектура программы**

#### **1. Главное окно (**MainWindow**)**

* Основной компонент программы.
* Отвечает за отображение базы данных, управление записями и взаимодействие с пользователем.
* Основные функции:
  + Создание, сохранение и загрузка базы данных.
  + Добавление, редактирование и удаление записей.
  + Поиск записей по критериям.
  + Обновление таблицы с данными.

#### **2. Класс данных (**PhoneModel**)**

* Реализует модель данных для хранения информации о смартфоне.
* Основные поля:
  + **Модель смартфона** (std::string model).
  + **Производитель** (std::string manufacturer).
  + **Операционная система** (std::string os).
  + **Объём памяти** (int memory).
  + **Цена** (double price).
* Основные методы:
  + Конструктор для создания объекта.
  + Геттеры и сеттеры для доступа к полям.

#### **3. Контейнер данных (**PhoneDatabase**)**

* Управляет коллекцией объектов PhoneModel.
* Использует контейнер std::vector.
* Основные методы:
  + Добавление нового смартфона (addPhone).
  + Редактирование записи (editPhone).
  + Удаление записи (removePhone).
  + Поиск записей (findPhones).
  + Сохранение данных в файл и загрузка из файла.

#### **4. Диалоговое окно (**PhoneEditor**)**

* Предоставляет интерфейс для ввода данных о смартфоне.
* Включает поля ввода для:
  + Модели смартфона.
  + Производителя.
  + Операционной системы.
  + Объёма памяти.
  + Цены.
* Выполняет проверку валидности данных перед их передачей в базу.

### **Графическая структура программы**

1. Таблица, отображающая текущую базу данных смартфонов.
2. Поле поиска и кнопка для выполнения поиска.
3. Кнопки для управления базой данных:
   * Добавление записи.
   * Редактирование записи.
   * Удаление записи.
   * Сохранение в файл.
   * Загрузка из файла.
   * "О программе".

Диалоговое окно (PhoneEditor) предоставляет форму для ввода данных. Кнопка "Добавить" становится активной только при вводе валидных данных.

# **ВЫБОР И ОБОСНОВАНИЕ ИСПОЛЬЗУЕМЫХ КЛАССОВ**

Для реализации функциональности программы были выбраны следующие ключевые классы: MainWindow, PhoneModel, PhoneDatabase и PhoneEditor. Каждый из них отвечает за определённую часть работы программы, обеспечивая модульность и удобство дальнейшей поддержки. В этом разделе обосновывается необходимость каждого из классов.

**4.1 MainWindow**

**Назначение**:  
Класс представляет главное окно программы. Он управляет пользовательским интерфейсом и взаимодействует с другими модулями для выполнения основных операций: добавления, редактирования, удаления записей, поиска данных, сохранения и загрузки базы данных.

**Обоснование**:

* Центральное окно нужно для отображения данных в табличной форме с помощью QTableWidget.
* Реализация функций кнопок (добавление, редактирование, удаление) и полей поиска невозможна без основного компонента управления интерфейсом.
* Интеграция с модулем работы с файлами (сохранение и загрузка) и контейнером данных осуществляется через MainWindow.

**4.2 PhoneModel**

**Назначение**:  
Класс предназначен для представления данных о смартфонах. Каждый объект класса соответствует одной записи в базе данных.

**Обоснование**:

* Использование отдельного класса для модели данных (PhoneModel) позволяет структурировать информацию о смартфоне в одном месте.
* Класс содержит поля, соответствующие основным характеристикам смартфона (модель, производитель, ОС, память, цена), а также методы доступа (геттеры и сеттеры).
* Такой подход упрощает взаимодействие с данными, предоставляя возможность их модификации или сериализации в файл.

**4.3 PhoneDatabase**

**Назначение**:  
Класс реализует контейнер для хранения и управления коллекцией объектов PhoneModel.

**Обоснование**:

* Хранение данных в контейнере std::vector позволяет эффективно управлять коллекцией записей.
* Предоставляет методы для выполнения базовых операций:
  + **Добавление записи**: метод addPhone позволяет легко добавлять новые записи.
  + **Редактирование записи**: метод editPhone обновляет существующие данные.
  + **Удаление записи**: метод removePhone удаляет данные по индексу.
  + **Поиск записей**: метод findPhones выполняет фильтрацию данных по ключевым словам.
* Класс отвечает за интеграцию с файловой системой (методы saveToFile и loadFromFile).

**4.4 PhoneEditor**

**Назначение**:  
Класс представляет диалоговое окно для ввода и редактирования данных о смартфоне.

**Обоснование**:

* Разделение интерфейса редактирования данных на отдельный класс позволяет улучшить модульность программы.
* Использование полей ввода (модель, производитель, ОС, память, цена) и кнопок подтверждения/отмены.
* Проверка введённых данных перед их сохранением упрощает контроль над корректностью базы данных.

**4.5 Связь классов**

**Взаимодействие между классами**:

* MainWindow управляет процессом взаимодействия пользователя с программой.
* PhoneDatabase хранит данные и предоставляет методы для выполнения операций с ними.
* PhoneModel представляет объект данных.
* PhoneEditor предоставляет интерфейс для ввода данных о смартфонах.

# **РАЗРАБОТКА СОБСТВЕННЫХ КЛАССОВ**

Для управления базой данных смартфонов в программе были разработаны следующие собственные классы: **PhoneModel**, **PhoneDatabase**, **PhoneManager**, **PhoneEditorDialog**. Каждый класс выполняет свою роль, обеспечивая структурированность и функциональность программы.

### **1. PhoneModel**

**Назначение**:  
PhoneModel описывает данные о смартфоне, которые сохраняются в базе данных. Каждый объект класса представляет одну запись.

**Поля**:

* **Модель** — название модели смартфона.
* **Производитель** — фирма-изготовитель устройства.
* **Операционная система** — тип операционной системы (Android, iOS и др.).
* **Память** — объём памяти устройства в гигабайтах.
* **Цена** — стоимость смартфона.

**Методы**:

* Конструкторы для создания объекта.
* Геттеры для получения значений полей.
* Сеттеры для изменения значений полей.

### **2. PhoneDatabase**

**Назначение**:  
Класс PhoneDatabase отвечает за хранение и управление коллекцией объектов PhoneModel. Это контейнер для работы с данными.

**Поля**:

* Коллекция смартфонов на основе контейнера std::vector.

**Методы**:

* **Добавление записи** — добавление нового смартфона в базу данных.
* **Удаление записи** — удаление существующего смартфона по индексу.
* **Редактирование записи** — изменение данных смартфона в указанной позиции.
* **Поиск записей** — поиск смартфонов по ключевому слову (например, модели или производителю).
* **Сохранение данных** — запись всей базы данных в файл.
* **Загрузка данных** — чтение базы данных из файла.

### **3. PhoneManager**

**Назначение**:  
Класс PhoneManager предоставляет утилитарные функции для работы с файлами базы данных.

**Методы**:

* **Создание новой базы данных** — создание пустого файла базы данных.
* **Сохранение базы данных** — запись всех записей из контейнера PhoneDatabase в файл.
* **Загрузка базы данных** — чтение данных из файла и заполнение контейнера.
* **Объединение баз данных** — слияние двух контейнеров PhoneDatabase в один.

### **4. PhoneEditorDialog**

**Назначение**:  
Класс представляет собой диалоговое окно для добавления и редактирования записей. Позволяет пользователю вводить или изменять данные о смартфонах.

**Элементы интерфейса**:

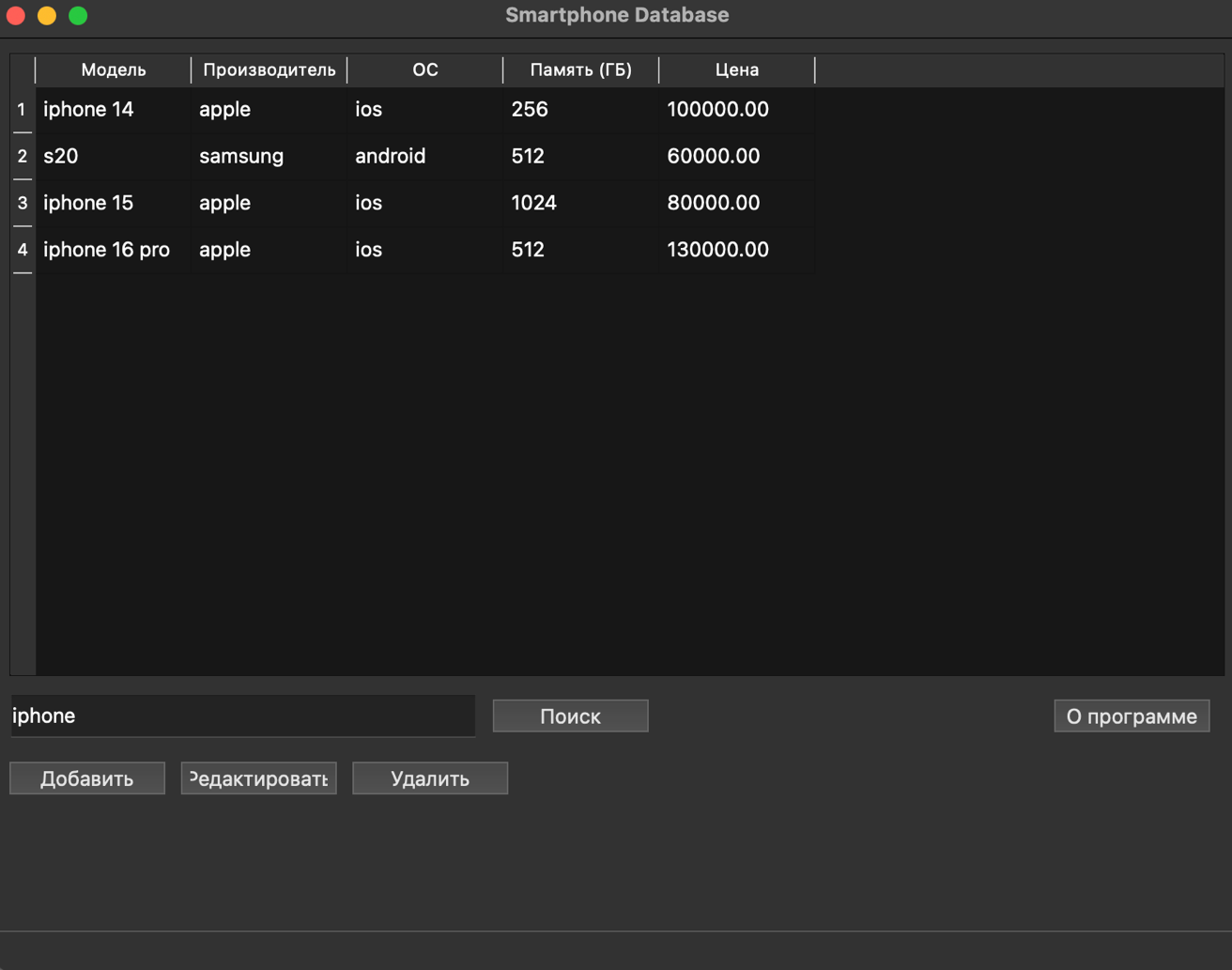
* Поля ввода:
  + Название модели смартфона.
  + Производитель устройства.
  + Операционная система.
  + Объём памяти (в ГБ).
  + Цена смартфона.
* Кнопки:
  + Подтверждение (Добавить или Сохранить).
  + Отмена (Отмена).

**Методы**:

* Установка данных для редактирования.
* Получение данных, введённых пользователем.
* Проверка корректности введённых значений.

# **РАЗРАБОТКА ИНТЕРФЕЙСА ПРОГРАММЫ**

Интерфейс программы разработан с учётом удобства работы пользователя и обеспечивает интуитивное взаимодействие с базой данных смартфонов. Он предоставляет функции добавления, редактирования, удаления записей, а также поиска данных и сохранения их в файл.



### **Основное окно (**MainWindow**)**

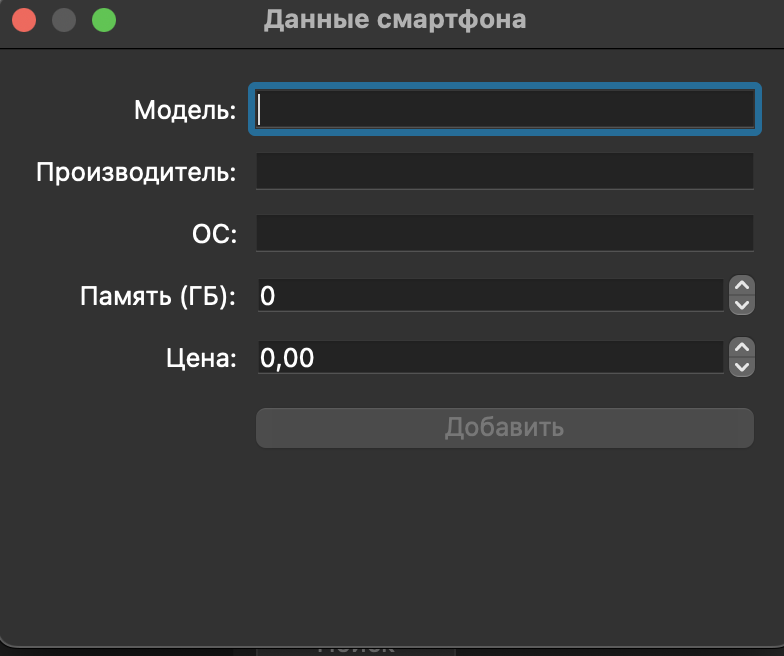
**Описание**:  
Главное окно приложения — это основной элемент взаимодействия пользователя с программой. Оно содержит таблицу для отображения данных, кнопки управления и поле для поиска.

**Элементы интерфейса**:

1. **Таблица данных**:
   * Отображает записи о смартфонах в табличной форме.
   * Столбцы таблицы:
     + Модель.
     + Производитель.
     + Операционная система.
     + Память (ГБ).
     + Цена.
   * Поддерживает выделение строк для редактирования или удаления записей.
2. **Кнопки управления**:
   * **Добавить**: открывает диалоговое окно для ввода новой записи.
   * **Редактировать**: позволяет изменить данные выбранной строки.
   * **Удалить**: удаляет выделенную строку.
   * **Сохранить**: сохраняет текущую базу данных в файл.
   * **Загрузить**: загружает данные из файла.
   * **О программе**: отображает информацию о приложении.
3. **Поле поиска**:
   * Поле ввода для текста, используемого для поиска записей.
   * Кнопка **"Поиск"** запускает фильтрацию записей по введённому ключевому слову.

**Особенности реализации**:

* Таблица обновляется автоматически при добавлении, редактировании или удалении записей.
* Функция поиска подсвечивает строки, соответствующие запросу.



### **Диалоговое окно (**PhoneEditorDialog**)**

**Описание**:  
Окно добавления и редактирования записей представляет собой отдельное диалоговое окно, где пользователь может вводить данные о смартфоне.

**Элементы интерфейса**:

1. Поля ввода:
   * **Название модели**: текстовое поле для ввода модели смартфона.
   * **Производитель**: текстовое поле для ввода названия компании.
   * **Операционная система**: текстовое поле для указания ОС (Android, iOS и др.).
   * **Память (ГБ)**: числовое поле для ввода объёма памяти.
   * **Цена**: числовое поле для ввода стоимости устройства.
2. Кнопки:
   * **Добавить** / **Сохранить**: сохраняет данные, введённые в поля.
   * **Отмена**: закрывает окно без сохранения изменений.

**Особенности реализации**:

* Кнопка подтверждения становится активной только при заполнении всех полей.
* Валидация данных предотвращает добавление некорректных записей.

### **Структура интерфейса**

1. **Главное окно**:
   * Поле поиска и кнопка "Поиск".
   * Таблица с данными.
   * Кнопки управления:
     + Добавить.
     + Редактировать.
     + Удалить.
     + Сохранить.
     + Загрузить.
     + О программе.
2. **Диалоговое окно**:
   * Поля для ввода данных.
   * Кнопки подтверждения и отмены.

# **ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЮ**

Программа управления базой данных смартфонов предоставляет интуитивно понятный интерфейс и широкий функционал для работы с данными. В этом разделе подробно описано, как использовать программу.

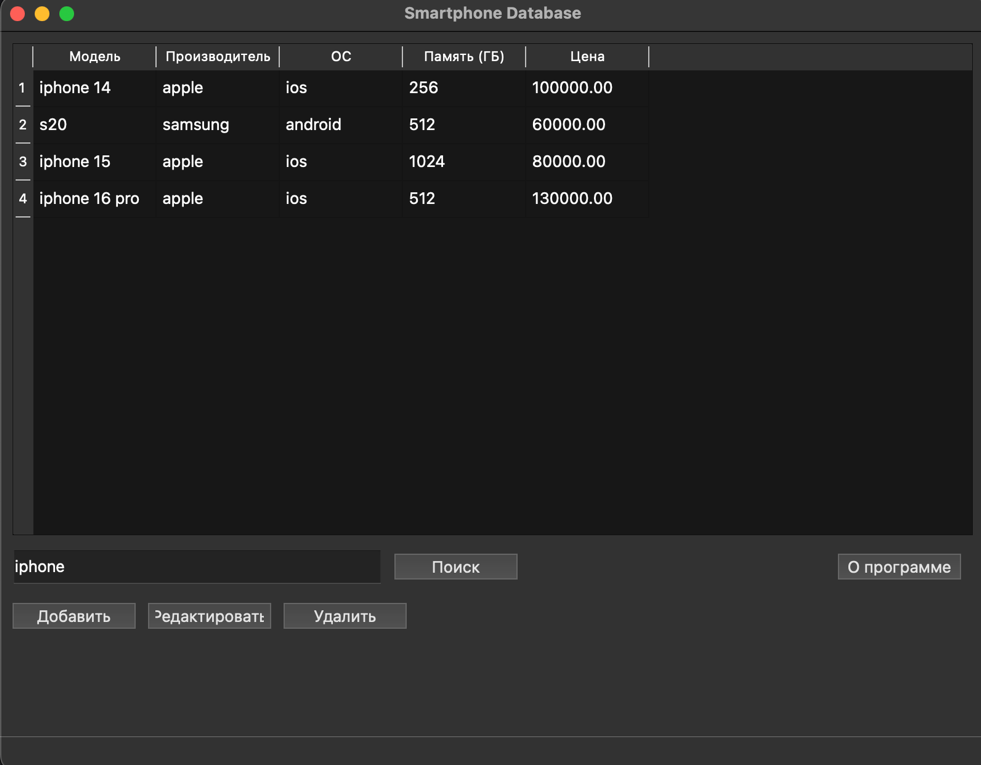
### **Запуск программы**

1. Запустите приложение, дважды щёлкнув по исполняемому файлу.
2. На экране появится главное окно программы.

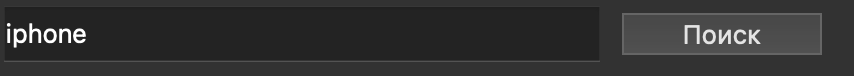


### **Элементы интерфейса**

1. **Таблица данных**:
   * Отображает текущую базу данных.
   * Столбцы таблицы:
     + **Модель** — название модели смартфона.
     + **Производитель** — название компании-изготовителя.
     + **Операционная система** — тип ОС (Android, iOS и т.д.).
     + **Память (ГБ)** — объём памяти.
     + **Цена** — стоимость устройства.

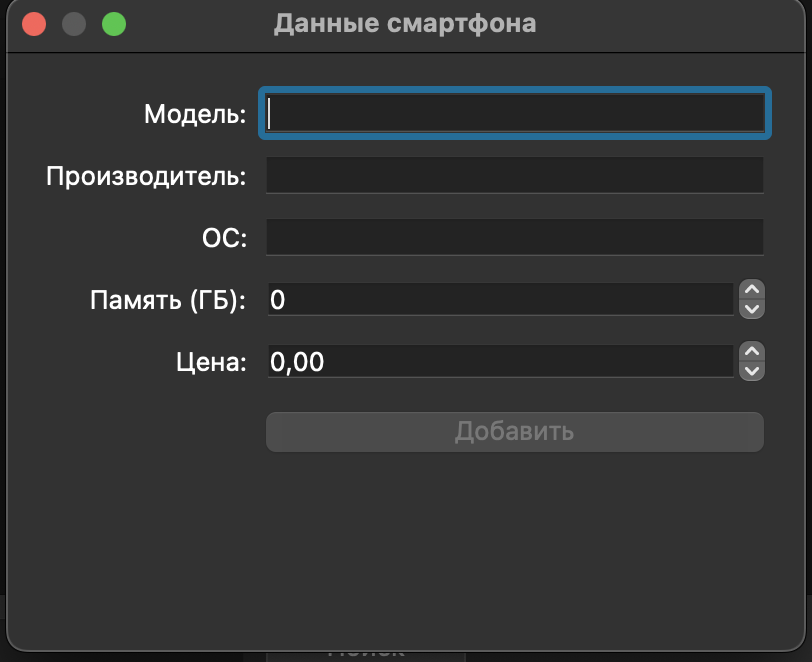


1. **Кнопки управления**:
   * **Добавить**: открыть диалоговое окно для добавления новой записи.
   * **Редактировать**: изменить данные выделенной строки.
   * **Удалить**: удалить выбранную строку.
   * **О программе**: отобразить информацию о приложении.
2. **Поле поиска**:
   * Поле ввода текста для поиска.
   * Кнопка **"Поиск"** подсвечивает записи, соответствующие введённому ключевому слову.



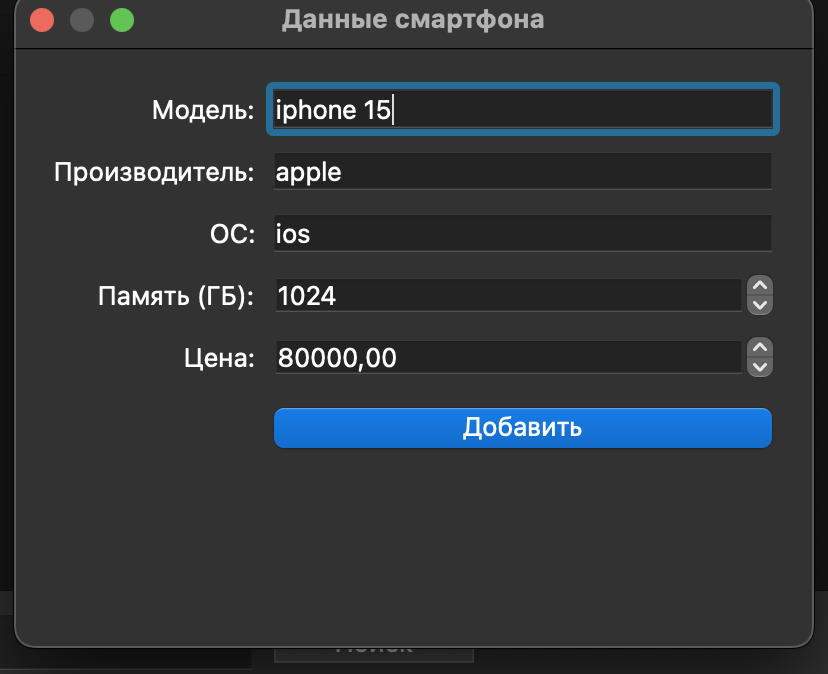
### **Работа с программой**

#### **Добавление записи**

1. Нажмите кнопку **"Добавить"**.
2. В открывшемся окне заполните поля:
   * **Модель**: название модели смартфона.
   * **Производитель**: компания-изготовитель.
   * **Операционная система**: тип ОС.
   * **Память (ГБ)**: укажите объём памяти.
   * **Цена**: введите стоимость устройства.
3. Нажмите кнопку **"Добавить"** для сохранения записи.

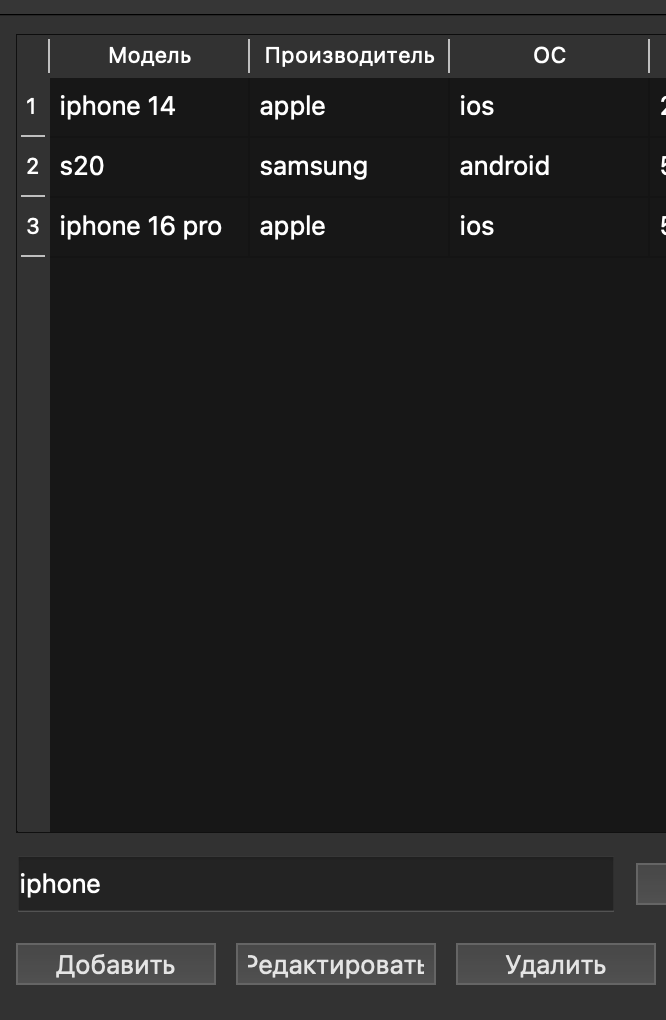
#### **Редактирование записи**

1. Выделите строку, которую хотите изменить.
2. Нажмите кнопку **"Редактировать"**.
3. В открывшемся окне измените данные в нужных полях.
4. Нажмите кнопку **"Сохранить"**.



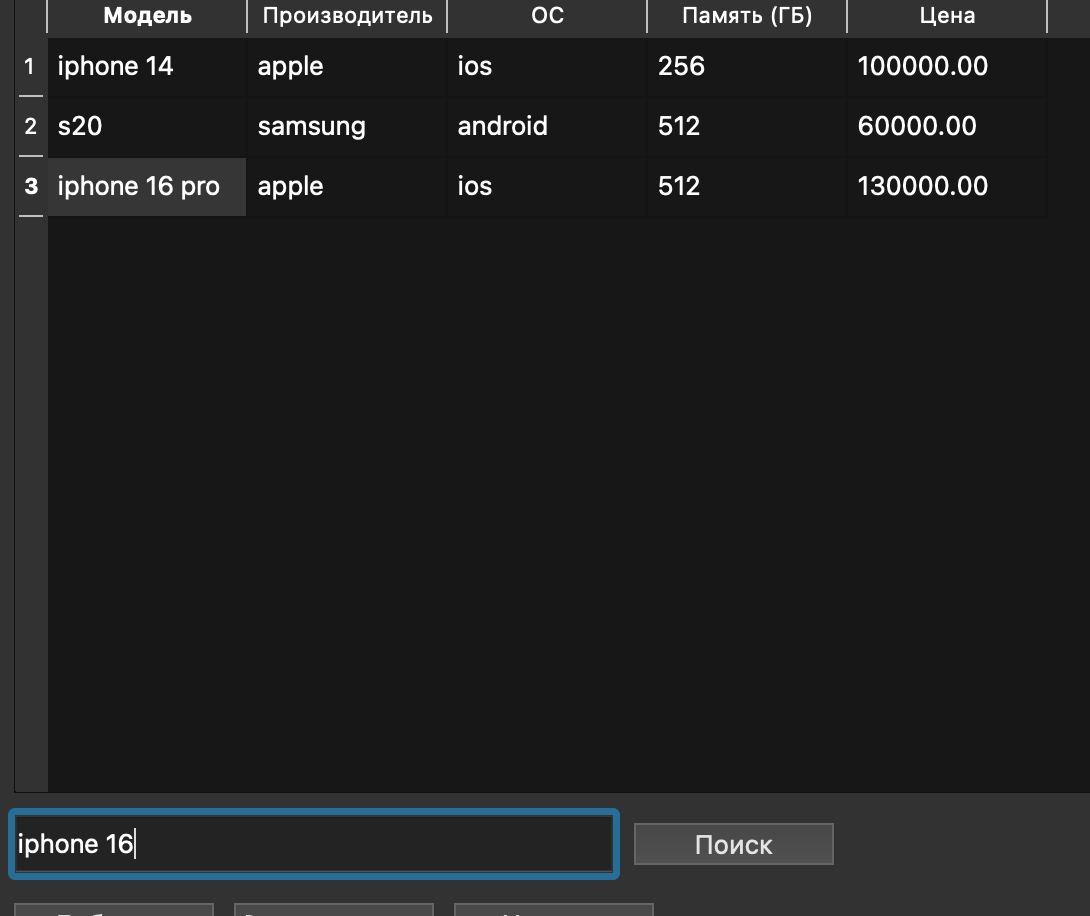
#### **Удаление записи**

1. Выделите строку, которую хотите удалить.
2. Нажмите кнопку **"Удалить"**.
3. Подтвердите удаление..



### **Поиск записей**

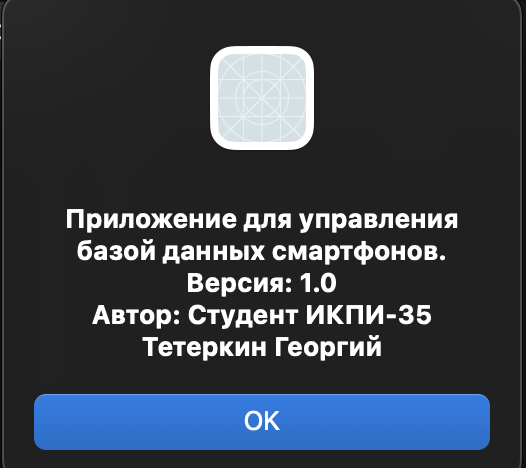
1. Введите текст в поле поиска (например, название модели или производителя).
2. Нажмите кнопку **"Поиск"**.
3. Программа подсветит записи, соответствующие запросу.



### **Дополнительные функции**

#### **О программе**

1. Нажмите кнопку **"О программе"**.
2. В появившемся окне вы увидите информацию о приложении, включая версию и автора.



# **ЗАКЛЮЧЕНИЕ**

Разработанная программа для управления базой данных смартфонов удовлетворяет всем заявленным требованиям. Она обеспечивает пользователю удобный интерфейс для работы с записями и предоставляет следующие основные функции:

1. **Создание новой базы данных** — программа позволяет создавать новую базу данных смартфонов с нуля.
2. **Добавление, редактирование и удаление записей** — интуитивный интерфейс позволяет легко управлять данными.
3. **Поиск данных** — реализована функция поиска записей по ключевым словам.
4. **Сохранение и загрузка базы данных** — данные можно сохранять в файл и загружать обратно для дальнейшей работы.
5. **Удобный пользовательский интерфейс**:
   * Главное окно для работы с базой данных.
   * Диалоговое окно для ввода или редактирования записей.
   * Таблица для наглядного отображения всех данных.

### **Достоинства программы**

* **Простота использования**: интерфейс программы интуитивно понятен даже для начинающих пользователей.
* **Модульная архитектура**: четкое разделение на классы упрощает сопровождение и расширение программы.
* **Функциональность**: программа поддерживает весь необходимый набор операций для работы с базой данных.
* **Переносимость**: использование библиотеки Qt делает программу кроссплатформенной.

### **Перспективы развития**

1. **Фильтрация данных**:
   * Добавление функции фильтрации по нескольким критериям, таким как цена или объём памяти.
2. **Экспорт данных**:
   * Возможность экспорта базы данных в формат CSV или JSON для интеграции с другими приложениями.
3. **Визуализация данных**:
   * Добавление графиков и статистики по данным, например, распределение цен или типов операционных систем.
4. **Мультипользовательский доступ**:
   * Реализация поддержки работы нескольких пользователей с базой данных через сеть.

### **Заключение**

Программа успешно реализует функционал управления базой данных смартфонов и соответствует современным требованиям к программному обеспечению. Её архитектура позволяет легко добавлять новые функции и адаптировать приложение к изменяющимся задачам.

# **СПИСОК ЛИТЕРАТУРЫ**

 **Бьёрн Страуструп** — "Программирование: принципы и практика с использованием C++". 2-е издание. — Москва: Бином, 2014.

 **Марк Саммерфилд** — "Программирование на Qt 5. Руководство для начинающих". — Санкт-Петербург: Питер, 2018.

 **Брюс Эккель** — "Философия C++. Введение в стандартный C++". Том 1. 4-е издание. — Москва: Вильямс, 2012.

 **Документация Qt** — https://doc.qt.io

 **Стандартная библиотека C++ (STL)** — https://en.cppreference.com

 **Уроки и статьи по Qt** — <https://habr.com/ru/hub/qt>

1. **ПРИЛОЖЕНИЕ**

### **MainWindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "PhoneDatabase.h"

#include "PhoneEditorDialog.h"

QT\_BEGIN\_NAMESPACE

namespace Ui {

class MainWindow;

}

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow {

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void createFile();

void saveFile();

void loadFile();

void mergeFiles();

void addPhone();

void editPhone();

void deletePhone();

void searchPhones();

void showAbout();

private:

Ui::MainWindow \*ui;

PhoneDatabase phoneDatabase;

QString currentDatabaseFile;

void updateTable();

int getSelectedRowIndex() const;

};

#endif // MAINWINDOW\_H

### **MainWindow.cpp**

#include "MainWindow.h"

#include "ui\_MainWindow.h"

#include <QFileDialog>

#include <QMessageBox>

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent), ui(new Ui::MainWindow) {

ui->setupUi(this);

ui->tableWidget->setColumnCount(5);

ui->tableWidget->setHorizontalHeaderLabels({"Модель", "Производитель", "ОС", "Память (ГБ)", "Цена"});

connect(ui->actionCreateFile, &QAction::triggered, this, &MainWindow::createFile);

connect(ui->actionSaveFile, &QAction::triggered, this, &MainWindow::saveFile);

connect(ui->actionLoadFile, &QAction::triggered, this, &MainWindow::loadFile);

connect(ui->actionMergeFiles, &QAction::triggered, this, &MainWindow::mergeFiles);

connect(ui->addButton, &QPushButton::clicked, this, &MainWindow::addPhone);

connect(ui->editButton, &QPushButton::clicked, this, &MainWindow::editPhone);

connect(ui->deleteButton, &QPushButton::clicked, this, &MainWindow::deletePhone);

connect(ui->searchButton, &QPushButton::clicked, this, &MainWindow::searchPhones);

connect(ui->aboutButton, &QPushButton::clicked, this, &MainWindow::showAbout);

}

MainWindow::~MainWindow() {

delete ui;

}

void MainWindow::createFile() {

QString filename = QFileDialog::getSaveFileName(this, tr("Создать файл базы данных"), "", tr("Database Files (\*.db);;All Files (\*)"));

if (!filename.isEmpty()) {

QFile file(filename);

if (file.open(QIODevice::WriteOnly)) {

currentDatabaseFile = filename;

QMessageBox::information(this, tr("Успех"), tr("Файл базы данных успешно создан."));

file.close();

} else {

QMessageBox::critical(this, tr("Ошибка"), tr("Ошибка при создании файла базы данных."));

}

}

}

void MainWindow::saveFile() {

if (currentDatabaseFile.isEmpty()) {

currentDatabaseFile = QFileDialog::getSaveFileName(this, tr("Сохранить файл"), "", tr("Database Files (\*.db);;All Files (\*)"));

if (currentDatabaseFile.isEmpty()) return;

}

QFile file(currentDatabaseFile);

if (file.open(QIODevice::WriteOnly)) {

QDataStream out(&file);

out << phoneDatabase.size();

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

PhoneModel phone = phoneDatabase.getPhone(i);

out << QString::fromStdString(phone.getModel())

<< QString::fromStdString(phone.getManufacturer())

<< QString::fromStdString(phone.getOS())

<< phone.getMemory()

<< phone.getPrice();

}

file.close();

QMessageBox::information(this, tr("Успех"), tr("Данные успешно сохранены."));

} else {

QMessageBox::critical(this, tr("Ошибка"), tr("Ошибка при сохранении файла."));

}

}

void MainWindow::loadFile() {

QString filename = QFileDialog::getOpenFileName(this, tr("Загрузить файл базы данных"), "", tr("Database Files (\*.db);;All Files (\*)"));

if (!filename.isEmpty()) {

QFile file(filename);

if (file.open(QIODevice::ReadOnly)) {

QDataStream in(&file);

int size;

in >> size;

phoneDatabase = PhoneDatabase();

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

QString model, manufacturer, os;

int memory;

double price;

in >> model >> manufacturer >> os >> memory >> price;

phoneDatabase.addPhone(PhoneModel(model.toStdString(), manufacturer.toStdString(), os.toStdString(), memory, price));

}

file.close();

updateTable();

QMessageBox::information(this, tr("Успех"), tr("Файл базы данных успешно загружен."));

} else {

QMessageBox::critical(this, tr("Ошибка"), tr("Ошибка при загрузке файла базы данных."));

}

}

}

void MainWindow::mergeFiles() {

QString filename = QFileDialog::getOpenFileName(this, tr("Выберите файл для объединения"), "", tr("Database Files (\*.db);;All Files (\*)"));

if (!filename.isEmpty()) {

QFile file(filename);

if (file.open(QIODevice::ReadOnly)) {

QDataStream in(&file);

int size;

in >> size;

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

QString model, manufacturer, os;

int memory;

double price;

in >> model >> manufacturer >> os >> memory >> price;

phoneDatabase.addPhone(PhoneModel(model.toStdString(), manufacturer.toStdString(), os.toStdString(), memory, price));

}

file.close();

updateTable();

QMessageBox::information(this, tr("Успех"), tr("Данные успешно объединены."));

} else {

QMessageBox::critical(this, tr("Ошибка"), tr("Ошибка при объединении файлов."));

}

}

}

void MainWindow::addPhone() {

PhoneEditorDialog dialog(this);

if (dialog.exec() == QDialog::Accepted) {

PhoneModel newPhone = dialog.getPhoneData();

phoneDatabase.addPhone(newPhone);

updateTable();

}

}

void MainWindow::editPhone() {

int selectedRow = getSelectedRowIndex();

if (selectedRow == -1) {

QMessageBox::warning(this, tr("Ошибка"), tr("Не выбрана строка для редактирования."));

return;

}

PhoneModel phone = phoneDatabase.getPhone(selectedRow);

PhoneEditorDialog dialog(this);

dialog.setFields(QString::fromStdString(phone.getModel()),

QString::fromStdString(phone.getManufacturer()),

QString::fromStdString(phone.getOS()),

phone.getMemory(),

phone.getPrice());

if (dialog.exec() == QDialog::Accepted) {

PhoneModel updatedPhone = dialog.getPhoneData();

phoneDatabase.editPhone(selectedRow, updatedPhone);

updateTable();

}

}

void MainWindow::deletePhone() {

int selectedRow = getSelectedRowIndex();

if (selectedRow == -1) {

QMessageBox::warning(this, tr("Ошибка"), tr("Не выбрана строка для удаления."));

return;

}

phoneDatabase.removePhone(selectedRow);

updateTable();

}

void MainWindow::searchPhones() {

QString keyword = ui->searchEdit->text().trimmed();

if (keyword.isEmpty()) {

QMessageBox::information(this, tr("Поиск"), tr("Введите ключевое слово для поиска."));

return;

}

for (int row = 0; row < ui->tableWidget->rowCount(); ++row) {

bool match = false;

for (int col = 0; col < ui->tableWidget->columnCount(); ++col) {

QTableWidgetItem\* item = ui->tableWidget->item(row, col);

if (item && item->text().contains(keyword, Qt::CaseInsensitive)) {

match = true;

break;

}

}

ui->tableWidget->setRowHidden(row, !match);

}

}

void MainWindow::showAbout() {

QMessageBox::information(this, tr("О программе"), tr("Программа для управления базой данных смартфонов.\nВерсия: 1.0\nАвтор: Ваше имя"));

}

void MainWindow::updateTable() {

ui->tableWidget->clearContents();

ui->tableWidget->setRowCount(phoneDatabase.size());

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

PhoneModel phone = phoneDatabase.getPhone(i);

ui->tableWidget->setItem(i, 0, new QTableWidgetItem(QString::fromStdString(phone.getModel())));

ui->tableWidget->setItem(i, 1, new QTableWidgetItem(QString::fromStdString(phone.getManufacturer())));

ui->tableWidget->setItem(i, 2, new QTableWidgetItem(QString::fromStdString(phone.getOS())));

ui->tableWidget->setItem(i, 3, new QTableWidgetItem(QString::number(phone.getMemory())));

ui->tableWidget->setItem(i, 4, new QTableWidgetItem(QString::number(phone.getPrice(), 'f', 2)));

}

}

int MainWindow::getSelectedRowIndex() const {

QList<QTableWidgetItem\*> selectedItems = ui->tableWidget->selectedItems();

if (selectedItems.isEmpty()) {

return -1;

}

return selectedItems.first()->row();

}

### **PhoneModel.h**

#ifndef PHONEMODEL\_H

#define PHONEMODEL\_H

#include <string>

class PhoneModel {

private:

std::string model;

std::string manufacturer;

std::string os;

int memory;

double price;

public:

PhoneModel(const std::string& model = "", const std::string& manufacturer = "",

const std::string& os = "", int memory = 0, double price = 0.0);

std::string getModel() const;

std::string getManufacturer() const;

std::string getOS() const;

int getMemory() const;

double getPrice() const;

void setModel(const std::string& model);

void setManufacturer(const std::string& manufacturer);

void setOS(const std::string& os);

void setMemory(int memory);

void setPrice(double price);

};

#endif // PHONEMODEL\_H

### **PhoneModel.cpp**

#include "PhoneModel.h"

PhoneModel::PhoneModel(const std::string& model, const std::string& manufacturer,

const std::string& os, int memory, double price)

: model(model), manufacturer(manufacturer), os(os), memory(memory), price(price) {}

std::string PhoneModel::getModel() const { return model; }

std::string PhoneModel::getManufacturer() const { return manufacturer; }

std::string PhoneModel::getOS() const { return os; }

int PhoneModel::getMemory() const { return memory; }

double PhoneModel::getPrice() const { return price; }

void PhoneModel::setModel(const std::string& model) { this->model = model; }

void PhoneModel::setManufacturer(const std::string& manufacturer) { this->manufacturer = manufacturer; }

void PhoneModel::setOS(const std::string& os) { this->os = os; }

void PhoneModel::setMemory(int memory) { this->memory = memory; }

void PhoneModel::setPrice(double price) { this->price = price; }

### **PhoneDatabase.h**

#ifndef PHONEDATABASE\_H

#define PHONEDATABASE\_H

#include "PhoneModel.h"

#include <vector>

#include <string>

class PhoneDatabase {

private:

std::vector<PhoneModel> phones;

public:

// Добавление записи

void addPhone(const PhoneModel& phone);

// Удаление записи

bool removePhone(int index);

// Редактирование записи

bool editPhone(int index, const PhoneModel& updatedPhone);

// Поиск записей по ключевому слову

std::vector<PhoneModel> findPhones(const std::string& keyword) const;

// Сохранение базы данных в файл

bool saveToFile(const std::string& filename) const;

// Загрузка базы данных из файла

bool loadFromFile(const std::string& filename);

// Получение количества записей

int size() const;

// Получение записи по индексу

PhoneModel getPhone(int index) const;

};

#endif // PHONEDATABASE\_H

### **PhoneDatabase.cpp**

#include "PhoneDatabase.h"

#include <fstream>

void PhoneDatabase::addPhone(const PhoneModel& phone) {

phones.push\_back(phone);

}

bool PhoneDatabase::removePhone(int index) {

if (index >= 0 && index < static\_cast<int>(phones.size())) {

phones.erase(phones.begin() + index);

return true;

}

return false;

}

bool PhoneDatabase::editPhone(int index, const PhoneModel& updatedPhone) {

if (index >= 0 && index < static\_cast<int>(phones.size())) {

phones[index] = updatedPhone;

return true;

}

return false;

}

std::vector<PhoneModel> PhoneDatabase::findPhones(const std::string& keyword) const {

std::vector<PhoneModel> results;

for (const auto& phone : phones) {

if (phone.getModel().find(keyword) != std::string::npos ||

phone.getManufacturer().find(keyword) != std::string::npos) {

results.push\_back(phone);

}

}

return results;

}

bool PhoneDatabase::saveToFile(const std::string& filename) const {

std::ofstream outFile(filename, std::ios::out);

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

return false;

}

for (const auto& phone : phones) {

outFile << phone.getModel() << ","

<< phone.getManufacturer() << ","

<< phone.getOS() << ","

<< phone.getMemory() << ","

<< phone.getPrice() << "\n";

}

outFile.close();

return true;

}

bool PhoneDatabase::loadFromFile(const std::string& filename) {

std::ifstream inFile(filename, std::ios::in);

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

return false;

}

phones.clear();

std::string model, manufacturer, os;

int memory;

double price;

while (inFile >> model >> manufacturer >> os >> memory >> price) {

phones.emplace\_back(model, manufacturer, os, memory, price);

}

inFile.close();

return true;

}

PhoneModel PhoneDatabase::getPhone(int index) const {

return phones.at(index);

}

int PhoneDatabase::size() const {

return static\_cast<int>(phones.size());

}

### **PhoneEditorDialog.h**

#ifndef PHONEEDITORDIALOG\_H

#define PHONEEDITORDIALOG\_H

#include <QDialog>

#include "PhoneModel.h"

namespace Ui {

class PhoneEditorDialog;

}

class PhoneEditorDialog : public QDialog {

Q\_OBJECT

public:

explicit PhoneEditorDialog(QWidget \*parent = nullptr);

~PhoneEditorDialog();

void setFields(const QString& model, const QString& manufacturer,

const QString& os, int memory, double price);

PhoneModel getPhoneData() const;

private slots:

void validateInput();

private:

Ui::PhoneEditorDialog \*ui;

bool validData;

};

#endif // PHONEEDITORDIALOG\_H

### **PhoneEditorDialog.cpp**

#include "PhoneEditorDialog.h"

#include "ui\_PhoneEditorDialog.h"

PhoneEditorDialog::PhoneEditorDialog(QWidget \*parent)

: QDialog(parent), ui(new Ui::PhoneEditorDialog), validData(false) {

ui->setupUi(this);

// Подключение сигналов для проверки ввода

connect(ui->lineEditModel, &QLineEdit::textChanged, this, &PhoneEditorDialog::validateInput);

connect(ui->lineEditManufacturer, &QLineEdit::textChanged, this, &PhoneEditorDialog::validateInput);

connect(ui->lineEditOS, &QLineEdit::textChanged, this, &PhoneEditorDialog::validateInput);

connect(ui->spinBoxMemory, &QSpinBox::valueChanged, this, &PhoneEditorDialog::validateInput);

connect(ui->doubleSpinBoxPrice, &QDoubleSpinBox::valueChanged, this, &PhoneEditorDialog::validateInput);

ui->pushButton\_add->setEnabled(false); // Кнопка изначально неактивна

}

PhoneEditorDialog::~PhoneEditorDialog() {

delete ui;

}

void PhoneEditorDialog::setFields(const QString& model, const QString& manufacturer,

const QString& os, int memory, double price) {

ui->lineEditModel->setText(model);

ui->lineEditManufacturer->setText(manufacturer);

ui->lineEditOS->setText(os);

ui->spinBoxMemory->setValue(memory);

ui->doubleSpinBoxPrice->setValue(price);

}

PhoneModel PhoneEditorDialog::getPhoneData() const {

return PhoneModel(

ui->lineEditModel->text().toStdString(),

ui->lineEditManufacturer->text().toStdString(),

ui->lineEditOS->text().toStdString(),

ui->spinBoxMemory->value(),

ui->doubleSpinBoxPrice->value()

);

}

void PhoneEditorDialog::validateInput() {

bool modelOk = !ui->lineEditModel->text().isEmpty();

bool manufacturerOk = !ui->lineEditManufacturer->text().isEmpty();

bool osOk = !ui->lineEditOS->text().isEmpty();

bool memoryOk = ui->spinBoxMemory->value() > 0;

bool priceOk = ui->doubleSpinBoxPrice->value() > 0;

validData = modelOk && manufacturerOk && osOk && memoryOk && priceOk;

ui->pushButton\_add->setEnabled(validData);

}

### **PhoneEditorDialog.ui**

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>PhoneEditorDialog</class>

<widget class="QDialog" name="PhoneEditorDialog">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>400</width>

<height>300</height>

</rect>

</property>

<property name="windowTitle">

<string>Редактирование смартфона</string>

</property>

<widget class="QWidget" name="formLayoutWidget">

<property name="geometry">

<rect>

<x>10</x>

<y>10</y>

<width>380</width>

<height>240</height>

</rect>

</property>

<widget class="QFormLayout" name="formLayout">

<item row="0" column="0">

<widget class="QLabel" name="labelModel">

<property name="text">

<string>Модель:</string>

</property>

</widget>

</item>

<item row="0" column="1">

<widget class="QLineEdit" name="lineEditModel"/>

</item>

<item row="1" column="0">

<widget class="QLabel" name="labelManufacturer">

<property name="text">

<string>Производитель:</string>

</property>

</widget>

</item>

<item row="1" column="1">

<widget class="QLineEdit" name="lineEditManufacturer"/>

</item>

<item row="2" column="0">

<widget class="QLabel" name="labelOS">

<property name="text">

<string>ОС:</string>

</property>

</widget>

</item>

<item row="2" column="1">

<widget class="QLineEdit" name="lineEditOS"/>

</item>

<item row="3" column="0">

<widget class="QLabel" name="labelMemory">

<property name="text">

<string>Память (ГБ):</string>

</property>

</widget>

</item>

<item row="3" column="1">

<widget class="QSpinBox" name="spinBoxMemory">

<property name="maximum">

<number>1024</number>

</property>

</widget>

</item>

<item row="4" column="0">

<widget class="QLabel" name="labelPrice">

<property name="text">

<string>Цена:</string>

</property>

</widget>

</item>

<item row="4" column="1">

<widget class="QDoubleSpinBox" name="doubleSpinBoxPrice">

<property name="maximum">

<double>999999.99</double>

</property>

<property name="decimals">

<number>2</number>

</property>

</widget>

</item>

</widget>

</widget>

<widget class="QPushButton" name="pushButton\_add">

<property name="geometry">

<rect>

<x>150</x>

<y>260</y>

<width>100</width>

<height>30</height>

</rect>

</property>

<property name="text">

<string>Добавить</string>

</property>

</widget>

</widget>

<resources/>

<connections/>

</ui>