МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский политехнический университет Петра Великого»

Институт компьютерных наук и кибербезопасности

Направление: 02.03.01 Математика и компьютерные науки

Отчет по дисциплине: «Научно-исследовательская работа»

«Работа в виртуальной среде»

Студент,
группы 5130201/40003 _____ Адиатуллин Т. Р.

Работу
принял _____ Глазунов В. В.

«____»_____ 2025 г.

Санкт-Петербург, 2025

Содержание

1. Введение

Современные информационные системы требуют эффективных инструментов для управления контактными данными. Телефонный справочник является базовым, но важным приложением, демонстрирующим принципы работы с данными, пользовательским интерфейсом и файловым хранилищем. В рамках данной лабораторной работы разработано консольное или десктопное приложение для управления телефонными контактами.

1.1. Используемые технологии

В ходе работы применяются следующие технологии и инструменты:

- **Язык программирования:** C++ (стандарт C++17)
- **Фреймворк разработки:** Qt Framework (версия 6.10)
- **Среда разработки:** Visual Studio Code
- **Система сборки:** CMake

1.2. Описание фреймворка Qt

Qt представляет собой кросс-платформенный фреймворк для разработки приложений с графическим интерфейсом пользователя. Фреймворк Qt обладает следующими ключевыми особенностями:

1.2.1. Механизм сигналов и слотов

Основой архитектуры Qt является механизм сигналов и слотов, который обеспечивает гибкую систему обмена сообщениями между объектами. Сигнал излучается объектом при наступлении определённого события, а слот представляет собой функцию, вызываемую в ответ на этот сигнал.

1.2.2. Виджеты и компоненты интерфейса

Qt предоставляет обширную библиотеку готовых виджетов для построения графического интерфейса:

- Контейнеры: QWidget, QMainWindow, QDialog
- Элементы ввода: QLineEdit, QTextEdit, QComboBox
- Кнопки: QPushButton, QRadioButton, QCheckBox
- Отображение данных: QTableWidget, QListWidget
- Диалоги: QMessageBox, QFileDialog, QInputDialog

1.2.3. Работа с данными

Qt включает мощные инструменты для работы с различными типами данных:

- Работа с файловой системой через QFile, QDir
- Поддержка XML (QXmlStreamReader/Writer) и JSON (QJsonDocument)
- Возможности работы с базами данных SQL через Qt SQL модуль
- Контейнеры данных: QList, QVector, QMap, QString

1.3. Предметная область — телефонный справочник

Телефонный справочник — это приложение для хранения, организации и управления контактной информацией. Типичная запись справочника содержит:

- **Личные данные:** фамилия, имя, отчество (опционально)
- **Контактная информация:** один или несколько номеров телефона
- **Дополнительные данные:** адрес электронной почты, физический адрес, примечания

Современный телефонный справочник должен обеспечивать следующие возможности:

1. **Управление записями:**
   - Добавление новых контактов с валидацией введённых данных
   - Просмотр полного списка сохранённых контактов
   - Редактирование существующих записей
   - Удаление контактов
2. **Поиск и фильтрация:**
   - Быстрый поиск по различным критериям (имя, фамилия, телефон, Email, дата рождения)
   - Поддержка частичного совпадения при поиске
3. **Сортировка данных:**
   - Упорядочивание по алфавиту
   - Сортировка по дате добавления записи

4. **Персистентность данных:**

   - Сохранение данных между сеансами работы приложения
   - Автоматическое или ручное сохранение изменений
   - Возможность экспорта и импорта данных

2. Постановка задачи

Цель работы — разработать функциональное приложение «Телефонный справочник» с графическим интерфейсом пользователя, реализующее полный набор операций по управлению контактными данными с соблюдением требований к валидации, хранению и отображению информации.

2.1. Формализованное задание

Разработать приложение «Телефонный справочник» со следующим функционалом:

2.1.1. Требования к функциональности

Управление контактами:

1. **Добавление нового контакта**
    - Ввод фамилии, имени, отчества контакта
    - Ввод номера телефона с валидацией формата
    - Ввод дополнительной информации (email, адрес)
    - Проверка уникальности номера телефона
    - Сохранение контакта в список
2. **Просмотр списка контактов**
    - Отображение всех контактов в виде таблицы или списка
    - Вывод информации: порядковый номер, фамилия, имя, телефон
    - Возможность сортировки по различным полям
3. **Поиск контакта**
    - Поиск по фамилии, имени, отчеству, email, дата рождения, номеру телефона
    - Отображение результатов поиска
    - Обработка случая отсутствия совпадений (пустота)
4. **Редактирование контакта**
    - Выбор существующего контакта для редактирования
    - Изменение любого поля записи
    - Повторная валидация изменённых данных

- Сохранение обновлённой информации

5. **Удаление контакта**

   - Выбор контакта для удаления
   - Подтверждение операции удаления
   - Удаление из списка и файла хранения

2.2. Технические требования

2.2.1. Формат хранения данных

Данные должны сохраняться в файл для обеспечения персистентности. Рекомендуемые форматы:

JSON формат

```
{
  "contacts": [
    {
      "id": 1,
      "firstName": "  ",
      "lastName": "   ",
      "phone": "+7 (123) 456-78-90",
      "email": "ivan.ivanov@example.com",
      "address": " .    ,  .     , . 1"
    },
    {
      "id": 2,
      "firstName": "   ",
      "lastName": "    ",
      "phone": "+7 (987) 654-32-10",
      "email": "maria.petrova@example.com",
      "address": ""
    }
  ]
}
```

2.2.2. Формат отображения данных

Контакты должны отображаться в удобном для пользователя виде:

2.2.3. Регулярные выражения для валидации

Для обеспечения корректности введённых данных необходимо применять следующие регулярные выражения:

Фамилия Имя Отчество

```
^[A-Za-z--0-9][A-Za-z--0-9\- ]*[A-Za-z--0-9]$
```

Требования: начинается с заглавной буквы, содержит только буквы и дефис (для двойных фамилий).

Номер телефона

```
^\+\d+$
```

Поддерживаемые форматы:

- +7 (123) 456-78-90
- +7 123 456 78 90
- 81234567890
- +79991234567

Email

```
//
^[A-Za-z0-9._%+\-]+$
//

   ^[A-Za-z0-9][A-Za-z0-9\-]*(\.[A-Za-z0-9][A-Za-z0-9\-]*)*\.[A-Za-z]{2,}$
```

Требования: стандартный формат электронной почты.

2.3. Дополнительные требования

2.3.1. Функции для обязательной реализации

Чтение и запись файла:

- `bool loadFromFile(const QString& filename)` — загрузка данных из файла при запуске
- `bool saveToFile(const QString& filename)` — сохранение данных в файл
- Обработка ошибок при работе с файлами (файл не существует, нет прав доступа)

Обработка пользовательского ввода:

- Валидация данных перед сохранением
- Очистка и нормализация введённых данных (удаление лишних пробелов)
- Информирование пользователя о некорректных данных
- Предотвращение ввода дубликатов номеров телефонов

Сортировка данных:

- Сортировка по фамилии (в алфавитном порядке)
- Сортировка по имени
- Возможность обратной сортировки
- Сохранение порядка сортировки между сеансами (опционально)

Поиск и фильтрация:

- Регистронезависимый поиск
- Поиск по части строки (подстроке)
- Множественные результаты при совпадении
- Быстрый доступ к найденным записям

2.4. Ограничения и особые требования

1. **Уникальность номеров:** В справочнике не может быть двух контактов с одинаковыми номерами телефонов
2. **Обязательные поля:** Фамилия, имя и телефон являются обязательными полями
3. **Кодировка:** Файл данных должен сохраняться в кодировке UTF-8 для корректной работы с кириллицей
4. **Резервное копирование:** При перезаписи файла желательно создавать резервную копию предыдущей версии
5. **Производительность:** Приложение должно корректно работать со справочником, содержащим до 1000 контактов

2.5. Ожидаемые результаты

По завершении работы должно быть получено:

- Полностью функциональное приложение с интерфейсом пользователя
- Исходный код, организованный в виде логически разделённых классов
- Файл данных с сохранёнными контактами

3. Реализация

3.1. Архитектура приложения

Приложение «Телефонный справочник» построено на основе объектно-ориентирован подхода с четким разделением ответственности между компонентами системы. Архитектура следует принципам модульности и слабой связанности, что обеспечивает удобство сопровождения и расширения функциональности.

Основные компоненты приложения:

- **Contact** — структура данных для хранения информации о контакте;
- **ContactValidator** — класс для валидации и нормализации пользовательского ввода;
- **ContactStorage** — класс для работы с файловым хранилищем (JSON);
- **MainWindow** — класс главного окна, реализующий графический интерфейс и логику взаимодействия с пользователем.

Структура директорий проекта:

```
lab8/
 CMakeLists.txt
 src/
    main.cpp
    contact.cpp
    contactstorage.cpp
    contactvalidator.cpp
    mainwindow.cpp
 headers/
     contact.h
     contactstorage.h
     contactvalidator.h
     mainwindow.h
```

Listing 1: Структура проекта

Взаимодействие компонентов представлено на диаграмме (см. рисунок 1).

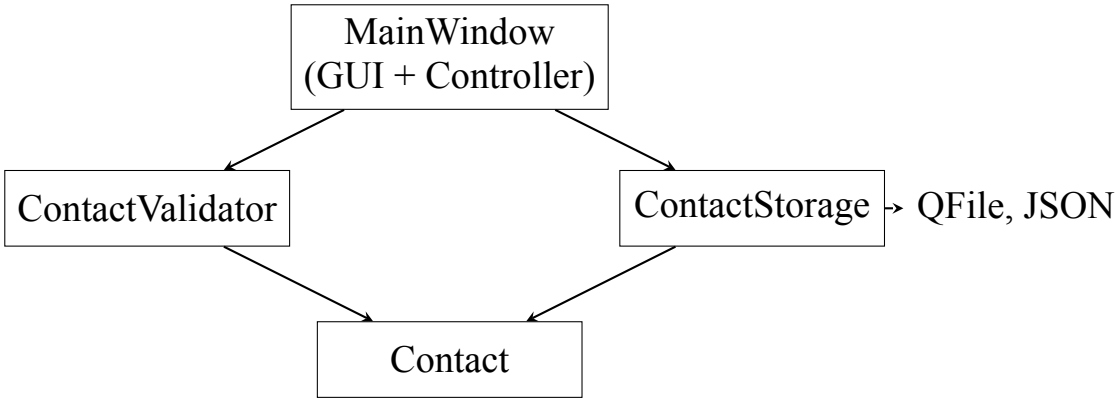

Рис. 1: Архитектура приложения

Класс MainWindow выступает центральным компонентом, координирующим работу остальных модулей. Он обрабатывает события пользовательского интерфейса, вызывает методы валидации перед операциями с данными и использует ContactStorage для сохранения и загрузки информации.

3.2. Описание классов

3.2.1. Структура Contact

Структура Contact представляет модель данных для хранения информации о контакте. Содержит все необходимые поля согласно требованиям задания.

```
#ifndef CONTACT_H
#define CONTACT_H

#include <QString>
#include <QDate>
#include <QStringList>
#include <QJsonObject>

class Contact
{
public:
    Contact();
    Contact(const QString& firstName, const QString& lastName,
            const QString& middleName, const QString& address,
            const QDate& birthDate, const QString& email,
            const QStringList& phoneNumbers);

    QString firstName() const { return m_firstName; }
    QString lastName() const { return m_lastName; }
    QString middleName() const { return m_middleName; }
    QString address() const { return m_address; }
    QDate birthDate() const { return m_birthDate; }
    QString email() const { return m_email; }
    QStringList phoneNumbers() const { return m_phoneNumbers; }

    void setFirstName(const QString& firstName);
    void setLastName(const QString& lastName);
    void setMiddleName(const QString& middleName);
    void setAddress(const QString& address);
    void setBirthDate(const QDate& birthDate);
    void setEmail(const QString& email);
    void setPhoneNumbers(const QStringList& phoneNumbers);

    QJsonObject toJson() const;
    static Contact fromJson(const QJsonObject& json);

    bool operator<(const Contact& other) const;
    bool operator==(const Contact& other) const;

private:
```

```
    QString m_firstName;
    QString m_lastName;
    QString m_middleName;
    QString m_address;
    QDate m_birthDate;
    QString m_email;
    QStringList m_phoneNumbers;
};

#endif // CONTACT_H
```

Listing 2: Заголовочный файл contact.h

Класс реализует методы сериализации в формат JSON и десериализации из него, что обеспечивает удобное сохранение данных в файл. Операторы сравнения используются для сортировки контактов по фамилии, имени и отчеству.

3.2.2. Класс ContactValidator

Класс `ContactValidator` инкапсулирует логику валидации пользовательского ввода с использованием регулярных выражений. Обеспечивает проверку корректности данных перед их сохранением.

```
#ifndef CONTACTVALIDATOR_H
#define CONTACTVALIDATOR_H

#include <QString>
#include <QDate>

class ContactValidator
{
public:
    static bool validateName(const QString& name,
                             QString& errorMessage);
    static bool validatePhone(const QString& phone,
                              QString& errorMessage);
    static bool validateBirthDate(const QDate& date,
                                  QString& errorMessage);
    static bool validateEmail(const QString& email,
                              QString& errorMessage);

    //
    static QString normalizeName(const QString& name);
    static QString normalizePhone(const QString& phone);
    static QString normalizeEmail(const QString& email);

private:
    static QString extractPhoneDigits(const QString& phone);
};

#endif // CONTACTVALIDATOR_H
```

Listing 3: Заголовочный файл contactvalidator.h

Регулярные выражения для валидации:

- **Имя:** `^[A-Za-z - - 0-9][A-Za-z - - 0-9\- ]*[A-Za-z - - 0-9]$`
  - Разрешены буквы (латиница и кириллица), цифры, дефис, пробел;
  - Не может начинаться или заканчиваться дефисом;
  - Запрещены двойные дефисы и множественные пробелы.
- **Телефон:** `^\+\d+$`
  - Международный формат (начинается с +);
  - Длина от 10 до 15 цифр;
  - Автоматическое преобразование формата 8(XXX)XXX-XX-XX в +7XXXXXXXX
- **Email:** `^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$`
  - Имя пользователя: латинские буквы, цифры, точки, подчеркивания;
  - Обязательное наличие символа @;
  - Домен должен содержать минимум одну точку.
- **Дата рождения:**
  - Должна быть меньше текущей даты;
  - Не может быть более 150 лет назад;
  - Валидация високосных годов выполняется классом QDate.

Псевдокод метода validateName:

Функция validateName(name, errorMessage):
normalized = normalizeName(name)
если normalized пусто:
errorMessage = ”Имя не может быть пустым”
вернуть false
trimmed = name.trimmed()
если trimmed начинается или заканчивается на ’-’:
errorMessage = ”Имя не может начинаться/заканчиваться на дефис”
вернуть false
regex = ^[A-Za-zA-Яа-яЁё0-9][A-Za-zA-Яа-яЁё0-9\-]*[A-Za-zA-Яа-яЁё0-9]$
если не соответствует regex:
errorMessage = ”Имя может содержать только буквы, цифры, дефис и пробел”
вернуть false

если содержит ”–” или множественные пробелы:
errorMessage = ”Имя не должно содержать двойные дефисы или множественные пробелы”
вернуть false
вернуть true

Методы нормализации:
Нормализация обеспечивает приведение данных к единому формату перед сохранением:

- `normalizeName` — удаляет лишние пробелы, приводит первую букву каждого слова к заглавной, остальные к строчным;
- `normalizePhone` — удаляет пробелы, скобки, дефисы, преобразует российский формат 8(...) в +7(...);
- `normalizeEmail` — приводит к нижнему регистру, удаляет пробелы.

3.2.3. Класс ContactStorage

Класс `ContactStorage` отвечает за сохранение и загрузку контактов из файла в формате JSON. Использует классы `QFile`, `QJsonDocument`, `QJsonArray` и `QJsonObject` из фреймворка Qt.

```
#ifndef CONTACTSTORAGE_H
#define CONTACTSTORAGE_H

#include "contact.h"
#include <QList>
#include <QString>

class ContactStorage
{
public:
    ContactStorage(const QString& filename = "phonebook.json");

    bool load();
    bool save();

    QList<Contact>& contacts();
    const QList<Contact>& contacts() const;

    void addContact(const Contact& contact);
    void removeContact(int index);
    void updateContact(int index, const Contact& contact);

    QString filename() const;
    void setFilename(const QString& filename);

private:
```

```
27      QList<Contact> m_contacts;
28      QString m_filename;
29  };
30
31  #endif // CONTACTSTORAGE_H
```

Listing 4: Заголовочный файл contactstorage.h

Псевдокод метода load:

Функция load():

```
file = открыть m_filename для чтения
если файл не существует:
    m_contacts.clear()
    вернуть true
если не удалось открыть файл:
    вывести ошибку
    вернуть false
data = прочитать весь файл
закрыть файл
doc = разобрать data как JSON
если ошибка парсинга:
    вывести ошибку
    вернуть false
если doc не является массивом:
    вывести ошибку
    вернуть false
array = doc.array()
m_contacts.clear()
для каждого value в array:
    если value является объектом:
        contact = Contact::fromJson(value)
        m_contacts.append(contact)
вернуть true
```

Псевдокод метода save:

Функция save():

```
file = открыть m_filename для записи
если не удалось открыть файл:
    вывести ошибку
    вернуть false
array = пустой JSON-массив
для каждого contact в m_contacts:
    jsonObject = contact.toJson()
    array.append(jsonObject)
```

doc = QJsonDocument(array)
записать doc.toJson() в файл
закрыть файл
вернуть true

3.2.4. Класс MainWindow

Класс `MainWindow` наследуется от `QMainWindow` и реализует графический интерфейс приложения, а также всю бизнес-логику взаимодействия с пользователем.

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTableWidget>
#include <QPushButton>
#include <QLineEdit>
#include <QDateEdit>
#include <QTextEdit>
#include <QComboBox>
#include <QStack>
#include "contactstorage.h"
#include "contact.h"

struct ContactAction {
    enum Type { Add, Edit, Delete };
    Type type;
    Contact contact;
    int index;
};

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void addContact();
    void editContact();
    void deleteContact();
    void searchContacts();
    void clearSearch();
    void onTableSelectionChanged();
    void loadContacts();
    void saveContacts();
    void undoLastAction();

private:
```

```
42      void setupUI();
43      void setupTable();
44      void setupForm();
45      void setupSearch();
46      void populateTable();
47      void clearForm();
48      bool validateForm(QString& errorMessage);
49      bool checkForDuplicates(const Contact& contact,
50                              int excludeIndex = -1);
51
52      // UI
53      QTableWidget* m_table;
54      QLineEdit* m_firstNameEdit;
55      QLineEdit* m_lastNameEdit;
56      QLineEdit* m_middleNameEdit;
57      QTextEdit* m_addressEdit;
58      QDateEdit* m_birthDateEdit;
59      QLineEdit* m_emailEdit;
60      QTextEdit* m_phoneNumbersEdit;
61      QPushButton* m_addButton;
62      QPushButton* m_editButton;
63      QPushButton* m_deleteButton;
64      QPushButton* m_undoButton;
65      QLineEdit* m_searchEdit;
66      QComboBox* m_searchFieldCombo;
67      QDateEdit* m_searchDateEdit;
68      QComboBox* m_dateSearchTypeCombo;
69
70      //
71      ContactStorage* m_storage;
72      int m_editingIndex;
73      bool m_isEditing;
74      QStack<ContactAction> m_undoStack;
75  };
76
77  #endif // MAINWINDOW_H
```

Listing 5: Фрагмент заголовочного файла mainwindow.h

Ключевые компоненты графического интерфейса:

- **QTableWidget** — таблица для отображения контактов с возможностью сортировки по столбцам;
- **QLineEdit, QTextEdit, QDateEdit** — поля ввода данных контакта;
- **QComboBox** — выпадающие списки для выбора поля поиска и типа сравнения даты;
- **QPushButton** — кнопки управления (Добавить, Редактировать, Удалить, Отмена, Поиск);
- **QGroupBox** — группировка элементов формы и поиска.

Внешний вид главного окна представлен на рисунке 2.

Рис. 2: Главное окно приложения

Механизм сигналов и слотов:

Qt использует архитектуру сигналов и слотов для обработки событий. Основные соединения в приложении:

```
//
connect(m_addButton, &QPushButton::clicked,
        this, &MainWindow::addContact);
connect(m_editButton, &QPushButton::clicked,
        this, &MainWindow::editContact);
connect(m_deleteButton, &QPushButton::clicked,
        this, &MainWindow::deleteContact);
connect(m_undoButton, &QPushButton::clicked,
        this, &MainWindow::undoLastAction);

//
connect(m_searchButton, &QPushButton::clicked,
        this, &MainWindow::searchContacts);
connect(m_clearSearchButton, &QPushButton::clicked,
        this, &MainWindow::clearSearch);

//
connect(m_table, &QTableWidget::itemSelectionChanged,
        this, &MainWindow::onTableSelectionChanged);

//
connect(m_storage, &ContactStorage::dataChanged,
        this, &MainWindow::saveContacts);
```

Listing 6: Соединения сигналов и слотов

3.3. Ключевые алгоритмы

3.3.1. Алгоритм добавления контакта

Процесс добавления нового контакта включает несколько этапов проверки и обработки данных. Псевдокод представлен ниже.

Псевдокод метода addContact:

Функция addContact():
errorMessage = ””
если не validateForm(errorMessage):
показать диалог с ошибкой errorMessage
вернуться
contact = получить данные из формы

```
если m_isEditing == true:
    oldContact = m_storage.contacts()[m_editingIndex]
    если checkForDuplicates(contact, m_editingIndex):
        показать диалог ”Такой контакт уже существует”
        вернуться
    m_storage.updateContact(m_editingIndex, contact)
    pushAction(Edit, oldContact, m_editingIndex)
    показать сообщение ”Контакт обновлен”
    m_isEditing = false
    m_addButton.setText(”Добавить”)
иначе:
    если checkForDuplicates(contact):
        показать предупреждение о дубликате
        если пользователь отменил:
            вернуться
    m_storage.addContact(contact)
    pushAction(Add, contact, m_storage.contactCount() - 1)
    показать сообщение ”Контакт добавлен”
m_storage.save()
populateTable()
clearForm()
```

3.3.2. Алгоритм поиска контактов

Приложение поддерживает два типа поиска:

- **Поиск по текстовым полям** (имя, фамилия, отчество, email, телефон, адрес) — поиск подстроки без учета регистра;
- **Поиск по дате рождения** — точное совпадение, диапазон ”до даты”, ”после даты”.

Псевдокод метода searchContacts:

Функция searchContacts():

```
searchField = m_searchFieldCombo.currentText()
m_table.clearContents()
m_table.setRowCount(0)
row = 0
если searchField == ”Дата рождения”:
    searchDate = m_searchDateEdit.date()
    searchType = m_dateSearchTypeCombo.currentText()
    для каждого contact в m_storage.contacts():
        match = false
        если searchType == ”Точное совпадение” и contact.birthDate() ==
```

searchDate:
match = true
если searchType == ”До даты” и contact.birthDate() <= searchDate:
match = true
если searchType == ”После даты” и contact.birthDate() >= searchDate:
match = true
если match:
добавить contact в таблицу, строка row
row = row + 1
иначе:
searchText = m_searchEdit.text().toLower()
для каждого contact в m_storage.contacts():
value = получить значение поля searchField из contact
если value содержит searchText (без учета регистра):
добавить contact в таблицу, строка row
row = row + 1
если row == 0:
показать сообщение ”Ничего не найдено”

3.3.3. Механизм отмены действий

Для реализации функции отмены последнего действия используется структура `ContactAction` и стек `QStack<ContactAction>`.

Структура ContactAction:

```
struct ContactAction {
    enum Type { Add, Edit, Delete };
    Type type;          //
    Contact contact;    //
    int index;          //       ( Edit   Delete)
};
```

Listing 7: Структура для хранения действия

Псевдокод метода undoLastAction:

Функция undoLastAction():
если m_undoStack пуст:
показать сообщение ”Нечего отменять”
вернуться
action = m_undoStack.pop()
если action.type == Add:
m_storage.removeContact(action.index)
показать сообщение ”Добавление отменено”
если action.type == Edit:
m_storage.updateContact(action.index, action.contact)
показать сообщение ”Редактирование отменено”

```
если action.type == Delete:
    m_storage.contacts().insert(action.index, action.contact)
    показать сообщение ”Удаление отменено”
m_storage.save()
populateTable()
clearForm()
```

При каждом изменении данных (добавление, редактирование, удаление) информация о действии сохраняется в стек методом `pushAction`. Это позволяет откатить последнюю операцию одним нажатием кнопки ”Отменить”.

3.3.4. Проверка на дубликаты

Перед добавлением или редактированием контакта выполняется проверка на наличие идентичных записей. Контакты считаются дубликатами, если совпадают все поля, кроме списка телефонов.

Псевдокод метода checkForDuplicates:

```
Функция checkForDuplicates(contact, excludeIndex):
    для i от 0 до m_storage.contactCount() - 1:
        если i == excludeIndex:
            продолжить следующую итерацию
        existing = m_storage.contacts()[i]
        если existing.firstName() == contact.firstName() и
                existing.lastName() == contact.lastName() и
                existing.middleName() == contact.middleName() и
                existing.birthDate() == contact.birthDate() и
                existing.email() == contact.email() и
                existing.address() == contact.address():
            вернуть true
    вернуть false
```

Если обнаружен дубликат, пользователю выводится п□□едупреждение с вопросом о продолжении добавления.

3.4. Сортировка данных

Сортировка контактов в таблице выполняется автоматически при клике на заголовок столбца благодаря встроенному механизму `QTableWidget`. Метод `setSortingE` активирует эту функциональность.

При каждой загрузке данных из файла список контактов предварительно сортируется по фамилии, имени и отчеству с помощью оператора `operator<`, реализованного в классе `Contact`:

```
bool Contact::operator<(const Contact& other) const
{
    if (m_lastName != other.m_lastName) {
        return m_lastName < other.m_lastName;
    }
    if (m_firstName != other.m_firstName) {
        return m_firstName < other.m_firstName;
    }
    return m_middleName < other.m_middleName;
}
```

Listing 8: Оператор сравнения для сортировки

Данный подход обеспечивает единообразие отображения данных при каждом запуске приложения.

4. Тестирование приложения

В данном разделе представлены основные сценарии использования приложения с описанием действий пользователя и ожидаемого поведения системы.

4.1. Начальное состояние приложения

При первом запуске приложения отображается главное окно с пустой таблицей контактов и формой ввода данных. Если файл `phonebook.json` не существует, он будет автоматически создан при сохранении первого контакта.

Рис. 3: Начальное состояние приложения при первом запуске

Элементы интерфейса:

- **Форма ввода данных** — содержит поля для всех атрибутов контакта;
- **Кнопки управления** — ”Добавить”, ”Редактировать”, ”Удалить”, ”Отменить”;
- **Панель поиска** — поле ввода, выбор поля для поиска, фильтры по дате;
- **Таблица контактов** — отображает список всех записей с возможностью сортировки;
- **Кнопки файловых операций** — ”Сохранить” и ”Загрузить”.

4.2. Сценарий 1: Добавление нового контакта

4.2.1. Шаг 1: Заполнение формы

Пользователь заполняет все поля формы корректными данными:

- **Фамилия:** Иванов
- **Имя:** Иван
- **Отчество:** Иванович
- **Адрес:** г. Москва, ул. Ленина, д. 1, кв. 10
- **Дата рождения:** 15.03.1990 (выбор через календарь)
- **Email:** ivan.ivanov@example.com
- **Телефоны:** +7 (999) 123-45-67, 8-800-555-35-35

Рис. 4: Заполненная форма для добавления контакта

4.2.2. Шаг 2: Нормализация данных

При вводе данных класс `ContactValidator` автоматически нормализует некоторые поля:

- **Имя ”иван”** преобразуется в **”Иван”** (первая буква заглавная);
- **Телефон ”8-800-555-35-35”** преобразуется в **”+78005553535”** (международный формат);
- **Email ”Ivan.Ivanov@Example.COM”** преобразуется в **”ivan.ivanov@example.com”** (нижний регистр).

Если происходит нормализация, пользователю выводится предупреждение с указанием изменений (см. рисунок 5).

Рис. 5: Предупреждение о нормализации данных

4.2.3. Шаг 3: Успешное добавление

После нажатия кнопки ”Добавить” происходит:

1. Валидация всех полей формы;
2. Проверка на дубликаты;
3. Добавление контакта в `ContactStorage`;
4. Автоматическое сохранение в файл `phonebook.json`;
5. Отображение контакта в таблице;
6. Сохранение действия в стек отмены;
7. Очистка полей формы;
8. Показ информационного сообщения об успешном добавлении.

Рис. 6: Контакт успешно добавлен в таблицу

4.3. Сценарий 2: Валидация некорректных данных

4.3.1. Пример 1: Некорректное имя

Пользователь вводит имя, начинающееся с дефиса: ”-Петр”. Система выводит ошибку валидации (см. рисунок 7).

Рис. 7: Ошибка валидации имени

Сообщение об ошибке: ”Имя не может начинаться или заканчиваться на дефис”

4.3.2. Пример 2: Некорректный телефон

Пользователь вводит телефон без знака ”+”: ”79991234567”. Класс `ContactValidator` автоматически добавляет ”+” в начало, преобразуя номер в ”+79991234567”.

Если длина телефона меньше 10 цифр, выводится ошибка: ”Телефон должен содержать минимум 10 цифр”.

4.3.3. Пример 3: Некорректный email

Пользователь вводит email без символа ”@”: ”ivanovexample.com”. Система выводит ошибку:

Сообщение об ошибке: ”Email должен содержать символ @”

4.3.4. Пример 4: Некорректная дата рождения

Пользователь не сможет выбрать дату которая больше текущей

4.4. Сценарий 3: Редактирование существующего контакта

4.4.1. Шаг 1: Выбор контакта

Пользователь кликает на строку в таблице. При этом:

- Данные контакта автоматически загружаются в форму;
- Активируются кнопки ”Редактировать” и ”Удалить”;
- Строка в таблице подсвечивается.

4.4.2. Шаг 2: Переход в режим редактирования

После нажатия кнопки ”Редактировать”:

- Приложение переходит в режим редактирования (`m_isEditing = true`);
- Кнопка ”Добавить” меняет текст на ”Сохранить”;
- Кнопки ”Редактировать” и ”Удалить” становятся неактивными;
- Появляется кнопка ”Отмена”;
- Таблица блокируется (нельзя выбрать другую строку);
- Все поля формы доступны для изменения.

Рис. 8: Режим редактирования контакта

4.4.3. Шаг 3: Изменение данных и сохранение

Пользователь изменяет необходимые поля (например, номер телефона) и нажимает ”Сохранить”. Происходит:

1. Валидация изменённых данных;
2. Проверка на дубликаты (исключая текущий редактируемый контакт);
3. Обновление контакта в `ContactStorage`;
4. Сохранение старой версии контакта в стек отмены;
5. Автоматическое сохранение в файл;
6. Возврат в режим просмотра;
7. Показ сообщения об успешном редактировании.

Рис. 9: Контакт успешно отредактирован

4.4.4. Шаг 4: Отмена редактирования

При нажатии кнопки ”Отмена”:

- Все изменения откатываются;
- Приложение возвращается в режим просмотра;
- Поля формы очищаются;
- Таблица снова становится активной.

4.5. Сценарий 4: Удаление контакта

4.5.1. Шаг 1: Выбор контакта для удаления

Пользователь выбирает строку в таблице и нажимает кнопку ”Уд□□лить”.

4.5.2. Шаг 2: Подтверждение удаления

Система выводит диалоговое окно с запросом подтверждения (см. рисунок 10).

Рис. 10: Запрос подтверждения удаления контакта

Сообщение: ”Вы уверены, что хотите удалить контакт ’Иванов Иван Иванович’?”

4.5.3. Шаг 3: Удаление и сохранение

Если пользователь подтверждает удаление:

1. Контакт сохраняется в стек отмены;
2. Контакт удаляется из `ContactStorage`;
3. Автоматическое сохранение в файл;
4. Обновление таблицы;
5. Показ сообщения ”Контакт удалён”.

Рис. 11: Контакт удалён из таблицы

Если пользователь отменяет удаление, никаких изменений не происходит.

4.6. Сценарий 5: Поиск контактов

4.6.1. Поиск по текстовым полям

Пользователь вводит текст в поле поиска и выбирает поле для поиска из выпадающего списка (Имя, Фамилия, Отчество, Email, Телефон, Адрес).

Пример: Поиск по фамилии ”Иванов”

Система выполняет поиск подстроки без учета регистра. Все контакты, содержащие ”иванов”, ”Иванов”, ”ИВАНОВ” в поле фамилии, отображаются в таблице (см. рисунок 12).

Рис. 12: Результат поиска по фамилии

4.6.2. Поиск по дате рождения

При выборе поля ”Дата рождения”:

- Поле текстового ввода скрывается;
- Отображается `QDateEdit` для выбора даты;
- Отображается `QComboBox` с типами сравнения: ”Точное совпадение”, ”До даты”, ”После даты”.

Рис. 13: Панель поиска по дате рождения

Пример 1: Точное совпадение
Пользователь выбирает дату ”15.03.1990” и тип ”Точное совпадение”. В таблице отображаются только контакты, родившиеся именно в этот день.
Пример 2: До даты
Пользователь выбирает дату ”01.01.2000” и тип ”До даты”. В таблице отображаются все контакты, родившиеся до 1 января 2000 года включительно.

Рис. 14: Результат поиска по дате (до 01.01.2000)

4.6.3. Сброс поиска

При нажатии кнопки ”Сбросить поиск”:

- Поле поиска очищается;
- Таблица отображает все контакты;
- Восстанавливается исходная сортировка.

Если результаты поиска пусты, выводится сообщение ”Ничего не найдено”.

4.7. Сценарий 6: Сортировка данных

Пользователь кликает на заголовок любого столбца таблицы. Происходит сортировка по этому столбцу:

- Первый клик — сортировка по возрастанию;
- Второй клик — сортировка по убыванию;
- Третий клик — возврат к исходной сортировке.

Рис. 15: Таблица, отсортированная по дате рождения

Сортировка работает для всех столбцов:

- **Текстовые поля** — лексикографическая сортировка;
- **Дата рождения** — хронологическая сортировка;
- **Телефоны** — сортировка как строки.

4.8. Сценарий 7: Отмена последнего действия

4.8.1. Отмена добавления

Пользователь добавил контакт, но сразу нажал кнопку ”Отменить”. Происходит:

1. Извлечение последнего действия из стека (`m_undoStack.pop()`);
2. Определение типа действия (Add);
3. Удаление контакта из `ContactStorage`;
4. Автоматическое сохранение в файл;
5. Обновление таблицы;
6. Показ сообщения ”Добавление отменено”.

Рис. 16: Сообщение об отмене последнего действия

4.8.2. Отмена редактирования

Пользователь отредактировал контакт и нажал ”Отменить”. Система восстанавливает старую версию контакта из стека и сохраняет изменения.

4.8.3. Отмена удаления

Пользователь удалил контакт и нажал ”Отменить”. Система восстанавливает удалённый контакт на его прежнюю позицию в списке.

4.8.4. Попытка отмены при пустом стеке

Если стек отмены пуст, при нажатии кнопки ”Отменить” выводится сообщение: ”Нечего отменять”.

4.9. Сценарий 8: Проверка на дубликаты

4.9.1. Обнаружение дубликата

Пользователь пытается добавить контакт с данными, идентичными существующему (кроме телефонов). Система обнаруживает дубликат и выводит предупреждение (см. рисунок 17).

Рис. 17: Предупреждение о дубликате контакта

Сообщение: ”Контакт с такими данными уже существует. Продолжить добавление?”

4.9.2. Варианты действий

- **Пользователь нажимает ”Да”** — контакт добавляется, несмотря на дублирование;
- **Пользователь нажимает ”Нет”** — операция отменяется, форма остаётся заполненной.

Проверка на дубликаты учитывает следующие поля:

- Имя, Фамилия, Отчество
- Дата рождения
- Email
- Адрес

Телефоны не учитываются при проверке, поскольку у одного человека может быть несколько номеров.

4.10. Сценарий 9: Сохранение и загрузка данных

4.10.1. Автоматическое сохранение

После каждого изменения данных (добавление, редактирование, удаление) приложение автоматически вызывает метод `ContactStorage::save()`, который записывает все контакты в файл `phonebook.json`.

Формат файла (JSON):

```
[
    {
        "firstName": "  ",
        "lastName": "   ",
        "middleName": "    ",
```

```
        "address": " .    ,  .    , . 1,  . 10",
        "birthDate": "1990-03-15",
        "email": "ivan.ivanov@example.com",
        "phoneNumbers": ["+79991234567", "+78005553535"]
    },
    {
        "firstName": "  ",
        "lastName": "   ",
        "middleName": "    ",
        "address": " .       -,      .,  . 50",
        "birthDate": "1985-07-20",
        "email": "petr.petrov@example.com",
        "phoneNumbers": ["+79161234567"]
    }
]
```

Listing 9: Пример phonebook.json

4.10.2. Загрузка при запуске

При запуске приложения автоматически вызывается метод `ContactStorage::load()` который загружает данные из файла. Если файл не существует или содержит некорректные данные, приложение начинает с пустой базы.

4.10.3. Ручное сохранение и загрузка

Кнопки ”Сохранить файл” и ”Загрузить файл” позволяют пользователю вручную управлять файлом данных. При нажатии кнопки ”Загрузить файл” все несохранённые изменения будут потеряны (после подтверждения).

4.11. Итоги тестирования

Все сценарии использования приложения были успешно протестированы. Приложение корректно обрабатывает:

- Валидацию пользовательского ввода с помощью регулярных выражений;
- Нормализацию данных перед сохранением;
- Операции CRUD (создание, чтение, обновление, удаление);
- Поиск по текстовым полям и по дате рождения;
- Сортировку данных по любому столбцу;
- Отмену последнего действия;
- Проверку на дубликаты;

- Автоматическое сохранение в файл формата JSON.

Графический интерфейс отзывчив и интуитивно понятен. Все операции сопровождаются информационными сообщениями, что повышает удобство использования приложения.

Заключение

В ходе выполнения лабораторной работы было разработано приложение «Телефонный справочник» с графическим пользовательским интерфейсом, реализованное с использованием фреймворка Qt.

Реализованный функционал

В рамках работы был полностью реализован требуемый функционал:

- **Добавление контактов** — реализован ввод данных о контактах с валидацией всех полей и автоматической нормализацией введённых данных;
- **Редактирование контактов** — реализована возможность изменения всех полей существующих контактов с сохранением валидации;
- **Удаление контактов** — реализовано удаление контактов с подтверждением действия;
- **Поиск контактов** — реализован поиск по всем текстовым полям, а также специализированный поиск по дате рождения с поддержкой трёх режимов (точная дата, год, месяц и год);
- **Сортировка данных** — реализована сортировка по всем полям таблицы с возможностью изменения направления сортировки;
- **Отмена действий** — реализован механизм отмены последнего действия (добавление, редактирование, удаление) с использованием стека операций;
- **Проверка на дубликаты** — реализована защита от добавления контактов с идентичными данными;
- **Сохранение и загрузка данных** — реализовано автоматическое сохранение данных в файл формата JSON после каждой операции и загрузка при запуске приложения.

Валидация данных

Была реализована комплексная система валидации вводимых данных с использованием регулярных выражений:

- **Имя, фамилия, отчество** — проверка на соответствие требованиям (буквы, цифры, дефисы, пробелы), запрет начала и окончания на дефис, автоматическая нормализация регистра символов;

- **Телефонные номера** — проверка формата международного номера, автоматическое преобразование российских номеров, начинающихся с «8», в формат «+7», удаление лишних символов форматирования;
- **Email** — проверка корректности адреса электронной почты с валидацией имени пользователя и доменного имени, автоматическое удаление пробелов и приведение к нижнему регистру;
- **Дата рождения** — проверка корректности даты, контроль, что дата находится в прошлом и не превышает разумный диапазон (150 лет).

Система валидации предоставляет пользователю информативные сообщения об ошибках, указывающие на конкретное поле и причину отклонения введённых данных.

Изученные технологии

В процессе разработки были изучены и освоены следующие технологии и компоненты фреймворка Qt:

- **Система сигналов и слотов Qt** — механизм взаимодействия объектов, обеспечивающий слабую связанность компонентов интерфейса;
- **Классы контейнеров Qt** — использовались классы `QList` для хранения списка контактов, `QStringList` для списка телефонных номеров, `QStack` для реализации отмены действий;
- **Виджеты Qt** — освоена работа с классами `QMainWindow`, `QTableWidget`, `QLineEdit`, `QTextEdit`, `QDateEdit`, `QComboBox`, `QPushButton`, `QGroupBox`, `QMessageBox`;
- **Работа с JSON** — использованы классы `QJsonDocument`, `QJsonObject`, `QJsonArray` для сериализации и десериализации данных;
- **Работа с файлами** — использован класс `QFile` для чтения и записи данных в файловую систему;
- **Регулярные выражения** — использован класс `QRegularExpression` для валидации вводимых пользователем данных;
- **Работа с датами** — освоена работа с классом `QDate` и виджетом `QDateEdit` с всплывающим календарём;
- **Система компоновки** — использованы классы `QVBoxLayout` и `QHBoxLayout` для автоматической компоновки элементов интерфейса.

Архитектурные решения

При разработке приложения были применены принципы объектно-ориентированного программирования и разделения ответственности:

- **Разделение модели и представления** — класс `Contact` инкапсулирует данные контакта, класс `MainWindow` отвечает за отображение и взаимодействие с пользователем;
- **Инкапсуляция логики валидации** — класс `ContactValidator` содержит всю логику проверки и нормализации данных, что обеспечивает повторное использование и упрощает тестирование;
- **Изоляция работы с хранилищем** — класс `ContactStorage` инкапсулирует операции чтения и записи файлов, что позволяет легко изменить формат хранения данных без модификации остальных компонентов.

Такая архитектура обеспечивает хорошую расширяемость приложения и упрощает его сопровождение.

Инструменты разработки

Для разработки приложения использовались следующие инструменты:

- **Фреймворк:** Qt 6;
- **Язык программирования:** C++17;
- **Среда разработки:** Qt Creator;
- **Система сборки:** CMake;
- **Система контроля версий:** Git.

Полученные навыки

В результате выполнения лабораторной работы были получены следующие знания и навыки:

- Проектирование графических пользовательских интерфейсов с использованием фреймворка Qt;
- Работа с механизмом сигналов и слотов для обработки событий;
- Применение регулярных выражений для валидации и нормализации данных;

- Работа с форматом JSON для хранения структурированных данных;
- Организация кода в соответствии с принципами ООП;
- Использование контейнеров и алгоритмов стандартной библиотеки Qt;
- Создание интуитивно понятных пользовательских интерфейсов.

Разработанное приложение полностью соответствует поставленным требованиям и может быть использовано в качестве основы для более сложных систем управления контактной информацией.

Список литературы

[1] Qt Documentation. Официальная документация Qt.
URL: https://doc.qt.io/ (дата обращения: 05.01.2026)

[2] Signals & Slots | Qt Core 6.x.
URL: https://doc.qt.io/qt-6/signalsandslots.html (дата обращения: 05.01.2026)

[3] JSON Support in Qt | Qt Core 6.x.
URL: https://doc.qt.io/qt-6/json.html (дата обращения: 05.01.2026)

[4] QRegularExpression Class | Qt Core 6.x.
URL: https://doc.qt.io/qt-6/qregularexpression.html (дата обращения: 05.01.2026)

[5] Qt Widgets 6.x.
URL: https://doc.qt.io/qt-6/qtwidgets-index.html (дата обращения: 05.01.2026)

[6] Regular expressions library (since C++11) - cppreference.com.
URL: https://en.cppreference.com/w/cpp/regex (дата обращения: 05.01.2026)

Полный исходный код

power.h

```
#pragma once
#include <stdio.h>

unsigned long long power_iterative(int base, int exp);
unsigned long long power_recursive(int base, int exp);
```

power.c

```
#include "headers/power.h"
#include <stdio.h>

unsigned long long power_iterative(int base, int exp) {
    unsigned long long result = 1;
    unsigned long long b = base;
    while (exp > 0) {
        if (exp % 2 == 1) {
            result *= b;
        }
        b *= b;
        exp /= 2;
    }
    return result;
}

unsigned long long power_recursive(int base, int exp) {
    if (exp == 0) return 1;
    if (exp == 1) return base;
    unsigned long long half = power_recursive(base, exp / 2);
    if (exp % 2 == 0)
        return half * half;
    else
        return half * half * base;
}
```

main.c

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <time.h>

#define REPEATS 100000

#include "src/headers/power.h"

int main(int argc, char **argv) {
    if (argc < 3) {
        printf("Usage: %s <base> <exponent>\n", argv[0]);
        return 1;
    }

    int base = atoi(argv[1]);
    int exp = atoi(argv[2]);

    clock_t start, end;
    unsigned long long result_iter = 0, result_rec = 0;
    double iter_total = 0.0, rec_total = 0.0;

    start = clock();
    for (int i = 0; i < REPEATS; ++i)
        result_rec = power_recursive(base, exp);
    end = clock();
    rec_total = (double)(end - start) / CLOCKS_PER_SEC;

    start = clock();
    for (int i = 0; i < REPEATS; ++i)
        result_iter = power_iterative(base, exp);
    end = clock();
    iter_total = (double)(end - start) / CLOCKS_PER_SEC;

    printfТест(" на %d повторов:\n", REPEATS);
    printfРезультат(": %llu. Итеративно: %.6f секв ( среднем %.9f сек
            result_iter, iter_total, iter_total / REPEATS);
    printfРезультат(": %llu. Рекурсивно: %.6f секв ( среднем %.9f сек
            result_rec, rec_total, rec_total / REPEATS);

    return 0;
}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10)
```

```
project(power_prac C)

set(CMAKE_C_STANDARD 11)
include_directories(headers)

add_library(power_static STATIC src/power.c)
set_target_properties(power_static PROPERTIES OUTPUT_NAME "pow

add_library(power_shared SHARED src/power.c)
set_target_properties(power_shared PROPERTIES OUTPUT_NAME "pow

add_executable(main_static main.c)
target_link_libraries(main_static power_static)

add_executable(main_shared main.c)
target_link_libraries(main_shared power_shared)

add_executable(main_dl main_dl.c)
target_link_libraries(main_dl dl)

add_executable(znak_ncurses_app znak_ncurses_app.c src/znak.c)

target_include_directories(znak_ncurses_app PRIVATE src/headers

target_link_libraries(znak_ncurses_app ncurses z)


add_custom_target(db
  COMMAND $(PROJECT_SOURCE_DIR)/build/znak_ncurses_app
  WORKING_DIRECTORY ${CMAKE_SOURCE_DIR}
  COMMENT "running db"
)
```

main_dl.c

```
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
#include <time.h>

#define REPEATS 100000

typedef unsigned long long (*power_iter_t)(int, int);
```

```
typedef unsigned long long (*power_rec_t)(int, int);

int main(int argc, char **argv) {
    if (argc < 3) {
        printf("Usage: %s <base> <exponent>\n", argv[0]);
        return 1;
    }

    int base = atoi(argv[1]);
    int exp = atoi(argv[2]);

    void *handle = dlopen("./libpower.so", RTLD_LAZY);
    if (!handle) {
        fprintf(stderr, Ошибка" dlopen: %s\n", dlerror());
        return 1;
    }

    power_iter_t power_iterative = (power_iter_t)dlsym(handle,
    power_rec_t power_recursive = (power_rec_t)dlsym(handle, "p

    char *error;
    if ((error = dlerror()) != NULL) {
        fprintf(stderr, Ошибка" dlsym: %s\n", error);
        dlclose(handle);
        return 1;
    }

    clock_t start, end;
    unsigned long long result_iter = 0, result_rec = 0;
    double iter_total = 0.0, rec_total = 0.0;

    start = clock();
    for (int i = 0; i < REPEATS; ++i)
        result_rec = power_recursive(base, exp);
    end = clock();
    rec_total = (double)(end - start) / CLOCKS_PER_SEC;

    start = clock();
    for (int i = 0; i < REPEATS; ++i)
        result_iter = power_iterative(base, exp);
    end = clock();
    iter_total = (double)(end - start) / CLOCKS_PER_SEC;

    printfТест(" на %d повторов:\n", REPEATS);
```

```
    printfРезультат(": %llu. Итеративно (dlopen): %.6f секв ( среднем
%.9f сек)\n",
            result_iter, iter_total, iter_total / REPEATS);
    printfРезультат(": %llu. Рекурсивно (dlopen): %.6f секв ( среднем
%.9f сек)\n",
            result_rec, rec_total, rec_total / REPEATS);

    dlclose(handle);
    return 0;
}
```

znak.h

```
#pragma once

#include <stdint.h>
#include <stddef.h>

typedef enum {
    ARIES, TAURUS, GEMINI, CANCER,
    LEO, VIRGO, LIBRA, SCORPIO,
    SAGITTARIUS, CAPRICORN, AQUARIUS, PISCES
} ZODIAC;

typedef struct {
    uint8_t day   : 5;  // –031
    uint8_t month : 4;  // –012
    uint16_t year : 12; // –04095
} BIRTHDAYDATE;


typedef struct {
    char surname[32];
    char name[32];
    ZODIAC zodiac;
    BIRTHDAYDATE birth;
} ZNAK;


typedef struct {
    char signature[4];
    uint32_t transaction_id;
    uint32_t record_count;
```

```
    uint32_t crc32;
} DB_HEADER;


uint32_t calculate_crc32(ZNAK *records, size_t count);
```

znak.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <zlib.h>
#include "headers/znak.h"

uint32_t calculate_crc32(ZNAK *records, size_t count) {
    return crc32(0L, (const unsigned char *)records, count * si
}
```

znak_ncurses_app.c

```
#include <ncurses.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>
#include <zlib.h>
#include "src/headers/znak.h"

#define MAX_RECORDS 100
#define DB_PATH "znak.db"

ZNAK records[MAX_RECORDS];
DB_HEADER header;
int record_count = 0;
uint32_t current_tx_id = 1; // поумолчанию

void input_record() {
    ZNAK z;
    char tmp[50];
    int zodiac_choice;

    clear();
    mvprintw(1, 2, "Add new record (ZNAK)");
```

```
mvprintw(3, 4, "Surname: ");
echo();
getstr(z.surname);

mvprintw(4, 4, "Name: ");
getstr(z.name);

// День
mvprintw(5, 4, "Day (1-31): ");
getstr(tmp);
int day = atoi(tmp);

if (day < 1 || day > 31) {
    z.birth.day = 0;
    mvprintw(6, 10, "Invalid day, set to 0");
} else {
    z.birth.day = day;
}

// Месяц
mvprintw(7, 4, "Month (1-12): ");
getstr(tmp);
int month = atoi(tmp);
if (month < 1 || month > 12) {
    z.birth.month = 0;
    mvprintw(8, 10, "Invalid month, set to 0");
} else {
    z.birth.month = month;
}

mvprintw(9, 4, "Year (1900-2100): ");
getstr(tmp);
int year = atoi(tmp);
if (year < 1900 || year > 2100) {
    z.birth.year = 0;
    mvprintw(10, 10, "Invalid year, set to 0");
} else {
    z.birth.year = year;
}

mvprintw(12, 4, "Choose zodiac (0-11):");
mvprintw(13, 6, "0: Aries, 1: Taurus, 2: Gemini, 3: Cancer"
mvprintw(14, 6, "4: Leo, 5: Virgo, 6: Libra, 7: Scorpio");
```

```
    mvprintw(15, 6, "8: Sagittarius, 9: Capricorn, 10: Aquarius
    getstr(tmp);
    zodiac_choice = atoi(tmp);
    if (zodiac_choice < 0 || zodiac_choice > 11) {
        zodiac_choice = 0;
        mvprintw(16, 10, "Invalid zodiac, set to 0");
    }
    z.zodiac = (ZODIAC)zodiac_choice;

    noecho();
    records[record_count++] = z;

    mvprintw(18, 2, "Record added. Press any key...");
    getch();
    header.transaction_id = current_tx_id++;
}

void show_records() {
    clear();
    mvprintw(1, 2, "List of ZNAK records:");
    mvprintw(2, 2, "  # | Surname      | Name         | Birthdat
| Zodiac");

    for (int i = 0; i < record_count; ++i) {
        mvprintw(4 + i, 2, " %2d | %-11s | %-10s | %02d.%02d.%0
                 i + 1,
                 records[i].surname,
                 records[i].name,
                 records[i].birth.day,
                 records[i].birth.month,
                 records[i].birth.year,
                 records[i].zodiac);
    }

    mvprintw(6 + record_count, 2, "Press any key to return...")
    getch();
}

void find_by_surname() {
    char tmp[32];
    clear();
    mvprintw(1, 2, "Enter surname to search: ");
    echo();
    getstr(tmp);
```

```
    noecho();

    int found = 0;
    mvprintw(3, 2, "Results for surname '%s':", tmp);

    for (int i = 0; i < record_count; ++i) {
        if (strcmp(records[i].surname, tmp) == 0) {
            mvprintw(5 + found, 4, "%s %s, %02d.%02d.%04d, Zod
                     records[i].surname,
                     records[i].name,
                     records[i].birth.day,
                     records[i].birth.month,
                     records[i].birth.year,
                     records[i].zodiac);
            found++;
        }
    }

    if (!found) {
        mvprintw(5, 2, "No people found with surname '%s'.", tm
    }

    getch();
    header.transaction_id = current_tx_id++;
}

void save_to_file() {
    FILE *f = fopen(DB_PATH, "wb");
    if (!f) {
        mvprintw(10, 2, "Could not open file for writing.");
        return;
    }

    memcpy(header.signature, "adia", 4);
    header.transaction_id = ++current_tx_id;
    header.record_count = record_count;
    header.crc32 = calculate_crc32(records, record_count);

    fwrite(&header, sizeof(DB_HEADER), 1, f);
    fwrite(records, sizeof(ZNAK), record_count, f);
    fclose(f);

    mvprintw(10, 2, "File saved successfully.");
}
```

```
void load_from_file() {
    FILE *f = fopen(DB_PATH, "rb");
    if (!f) {
        mvprintw(10, 2, "File not found. Please enter new data.
        return;
    }

    fread(&header, sizeof(DB_HEADER), 1, f);
    fread(records, sizeof(ZNAK), header.record_count, f);
    record_count = header.record_count;
    current_tx_id = header.transaction_id + 1;
    fclose(f);

    mvprintw(10, 2, "Loaded %d records from file.", record_coun
}

void run_ui() {
    initscr();
    cbreak();
    noecho();

    int choice;
    while (true) {
        clear();
        mvprintw(1, 2, "ZNAK Menu:");
        mvprintw(3, 4, "1. Add new record");
        mvprintw(4, 4, "2. Show all records");
        mvprintw(5, 4, "3. Find by surname");
        mvprintw(6, 4, "4. Save to file");
        mvprintw(7, 4, "5. Load from file");
        mvprintw(8, 4, "6. Exit");

        mvprintw(10, 2, "Your choice: ");
        echo();
        scanw("%d", &choice);
        noecho();

        switch (choice) {
            case 1: input_record(); break;
            case 2: show_records(); break;
            case 3: find_by_surname(); break;
            case 4: save_to_file(); getch(); break;
            case 5: load_from_file(); getch(); break;
```

```
            case 6: endwin(); return;
            default: break;
        }
    }
}

int main() {
    run_ui();
    return 0;
}
```