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: «Дискретная математика»

«Реализация генерации бинарного кода Грея и операций над мультимножествами на его основе».

Студент,
группы 5130201/40003 _____ Селезнев К. Д.

Руководитель,
Преподаватель _____ Востров А.В.

«____»_____ 2025 г.

Санкт-Петербург, 2025

Содержание

1 Введение

В работе рассматривается разработка приложения «Телефонный справочник» с использованием фреймворка Qt. Приложение относится к классу систем управления данными и демонстрирует принципы объектно-ориентированного программирования, работу с пользовательским вводом и хранение информации в файле.

1.1 Фреймворк Qt

Qt представляет собой кроссплатформенную библиотеку для создания графических интерфейсов и работы с данными. В работе используются модули Qt Widgets для построения интерфейса, механизм сигналов и слотов для организации взаимодействия между объектами, а также классы QFile и QRegularExpression для работы с файлами и проверки корректности ввода. Дополнительно применяются средства для работы с датами и контейнерами, что позволяет реализовать хранение и обработку контактной информации.

1.2 Предметная область

Приложение «Телефонный справочник» предназначено для управления контактной информацией о пользователях. Каждый контакт включает имя, фамилию, отчество, адрес проживания, дату рождения, электронную почту и телефонные номера. Система должна обеспечивать добавление, редактирование и удаление записей, а также их поиск по выбранному столбцу и сортировку также по выбранному столбцу. Для хранения данных используется текстовый файл, а для представления информации пользователю — табличный интерфейс.

1.3 Цель работы

Целью работы является разработка приложения для управления контактами на базе фреймворка Qt. В процессе выполнения необходимо освоить работу с табличными представлениями данных, механизмом сигналов и слотов, средствами валидации пользовательского ввода и методами сохранения и загрузки информации из файла.

2 Постановка задачи

2.1 Требования к хранимым данным

Каждая запись в телефонном справочнике представляется объектом класса `Contact`, который содержит набор полей для хранения информации о пользователе. В состав данных входят фамилия, имя, отчество, адрес проживания, дата рождения, адрес электронной почты и список телефонных номеров. Для строковых данных используется тип `QString`, для даты рождения — тип `QDate`, для набора телефонов — контейнер `QStringList`.

Фамилия, имя и отчество должны начинаться с заглавной буквы, а последующие буквы должны быть маленькими и содержать только допустимые символы. Адрес проживания хранится в виде строки без ограничений на формат, но с удалением лишних пробелов и при наличии символа «|» также происходит его удаление, дабы не перечить загрузке контакта из файла, где используется этот разделительный символ. Дата рождения должна быть корректной и меньше текущей даты хотя бы на один день. Электронная почта должна включать имя пользователя, символ «@», доменное имя и доменную зону. Телефонные номера допускают национальный и международный формат, при сохранении они нормализуются до последовательности цифр и символа «+». Минимальная длина нормализованного номера составляет десять символов.

Все данные сохраняются в текстовом файле, где каждая строка соответствует одному контакту. Поля разделяются символом «|». Для корректной загрузки и сохранения запрещено использование разделителя внутри значений полей.

2.2 Функциональные требования

Приложение должно реализовывать следующие функциональные возможности.

Добавление нового контакта.

Пользователь должен иметь возможность добавить новый контакт в справочник. Форма ввода должна содержать поля для всех обязательных данных контакта. После успешного добавления контакт должен появиться в табличном представлении.

Редактирование существующего контакта.

Пользователь должен иметь возможность изменить данные выбранного контакта. После внесения изменений и их валидации обновленные данные долж-

ны отразиться в таблице.

Удаление контакта.

Пользователь должен иметь возможность удалить выбранный контакт из справочника. Кнопка удаления должна быть активна только при выбранной строке в таблице. После удаления контакт должен исчезнуть из табличного представления.

Поиск и фильтрация контактов.

Пользователь должен иметь возможность найти контакт по введённому запросу и выбранному полю. Поиск должен выполняться по каждому полю таблицы. Фильтрация должна происходить в реальном времени при вводе текста. При очистке поля поиска должны отображаться все контакты.

Сортировка данных.

Пользователь должен иметь возможность упорядочить контакты по любому полю. Сортировка должна выполняться через клик по заголовку столбца в таблице. При этом после сортировки должен корректно запоминаться индекс уже созданного контакта, чтобы впоследствии при редактировании и удалении выбирался нужный.

Сохранение и загрузка данных.

Данные справочника должны сохраняться на локальное устройство и загружаться. После каждой операции добавления, редактирования или удаления данных должно реализовываться корректное сохранение в файл, путь которого указан в программе.

2.3 Требования к валидации данных

Все вводимые пользователем данные должны проходить проверку на соответствие заданным правилам с использованием регулярных выражений.

Валидация имени, фамилии и отчества.

Длина строки должна быть больше нуля после удаления пробелов по краям. Первый символ должен быть заглавной буквой кириллицы или латиницы, последующие обязательно должны быть записаны с маленькой буквы. Допустимые символы включают буквы различных алфавитов, пробелы, дефисы и цифры. Строка не может начинаться или заканчиваться дефисом. Пробелы

в начале и конце строки должны автоматически удаляться.

Валидация телефонного номера.

Допускается международный формат с символом плюс или национальный формат. Допустимые символы включают цифры, символ плюс, пробелы. Примеры допустимых форматов: `+79287766000`, `89287766000`. При сохранении телефон нормализуется до последовательности цифр и символа плюс. Минимальная длина нормализованного номера составляет 10 символов.

Валидация даты рождения.

Дата должна быть валидной, что проверяется через QDate::isValid. Дата должна быть строго меньше текущей даты. Проверка выполняется сравнением с QDate::currentDate. Для ввода используется виджет QDateEdit с форматом отображения DD.MM.YYYY. Виджет имеет всплывающий календарь, активируемый через setCalendarPopup.

Валидация Email.

Формат должен соответствовать схеме username@domain.zone. Имя пользователя должно состоять из одной или более латинских букв или цифр. Обязательно наличие символа @. Доменное имя должно состоять из одной или более латинских букв или цифр. Обязательно наличие точки после домена. Зона домена должна содержать одну или более латинских букв или цифр. Пробелы в строке должны автоматически удаляться.

2.4 Требования к пользовательскому интерфейсу

Таблица должна поддерживать автоматическую сортировку по столбцам. Режим выбора должен позволять выделение только целых строк через SelectRows.

Форма ввода и редактирования реализуется через отдельное диалоговое окно ContactDialog, которое содержит все необходимые поля. Кнопки управления располагаются в нижней части главного окна. Состояние кнопок зависит от текущего контекста, например, кнопки редактирования, удаления активны только при выборе строки в таблице.

3 Реализация

3.1 Архитектура приложения

Приложение построено с использованием объектно-ориентированного подхода и разделения ответственности между компонентами. Структура проекта организована следующим образом: в директории `headers` располагаются заголовочные файлы классов: Contact.h для структуры данных контакта, ContactDialog.h для диалогового окна редактирования и PhoneBook.h для главного окна приложения. Директория `src` содержит файлы реализации соответствующих классов, а также файл main.cpp с точкой входа в программу. Файл project.pro описывает конфигурацию проекта для системы сборки qmake.

Архитектура приложения включает несколько ключевых компонентов. Класс Contact представляет собой модель данных для хранения информации об одном контакте. Класс PhoneBook является главным окном приложения и содержит таблицу для отображения всех контактов, а также кнопки управления. Класс ContactDialog реализует диалоговое окно для добавления и редактирования контактов с полной валидацией вводимых данных.

3.2 Описание классов

3.2.1 Класс Contact

Класс Contact инкапсулирует данные одного контакта. Он содержит приватные поля для хранения фамилии, имени, отчества, адреса, даты рождения, email и списка телефонов. Для каждого поля предоставлены геттеры и сеттеры, обеспечивающие контролируемый доступ к данным.

Конструктор по умолчанию инициализирует дату рождения текущей датой. Метод toString преобразует объект контакта в строковое представление для сохранения в файл, используя символ вертикальной черты как разделитель полей. Статический метод fromString выполняет обратную операцию, создавая объект Contact из строки.

```
Contact::Contact(){
    birthDate = QDate::currentDate();
}

QString Contact::getLastName() const{ return lastName; }
QString Contact::getFirstName() const{ return firstName; }
QString Contact::getMiddleName() const{ return middleName; }
QString Contact::getAddress() const{ return address; }
QDate Contact::getBirthDate() const{ return birthDate; }
QString Contact::getEmail() const{ return email; }
QStringList Contact::getPhones() const{ return phones; }
```

```
12
13 void Contact::setLastName(const QString &value){ lastName = value
   ; }
14 void Contact::setFirstName(const QString &value){ firstName =
   value; }
15 void Contact::setMiddleName(const QString &value){ middleName =
   value; }
16 void Contact::setAddress(const QString &value){ address = value;
   }
17 void Contact::setBirthDate(const QDate &value){ birthDate = value
   ; }
18 void Contact::setEmail(const QString &value){ email = value; }
19 void Contact::setPhones(const QStringList &value){ phones = value
   ; }
20
21 QString Contact::toString() const{
22     return lastName + "|" + firstName + "|" + middleName + "|" +
23     address + "|" + birthDate.toString("yyyy-MM-dd") + "|" +
24     email + "|" + phones.join(",");
25 }
26
27 Contact Contact::fromString(const QString &str){
28     Contact contact;
29     QStringList parts = str.split('|');
30
31     if (parts.size() == 7){
32         contact.setLastName(parts[0]);
33         contact.setFirstName(parts[1]);
34         contact.setMiddleName(parts[2]);
35         contact.setAddress(parts[3]);
36         contact.setBirthDate(QDate::fromString(parts[4], "yyyy-MM
              -dd"));
37         contact.setEmail(parts[5]);
38         contact.setPhones(parts[6].split(','));
39     }
40     return contact;
41 }
```

Листинг 1: Класс Contact

3.2.2 Класс ContactDialog

Класс `ContactDialog` наследуется от `QDialog` и реализует диалоговое окно для ввода и редактирования данных контакта. Конструктор создает форму с использованием `QFormLayout`, добавляя все необходимые поля ввода.

Для каждого типа данных используется соответствующий виджет. Поля имени, фамилии, отчества, адреса и email представлены объектами `QLineEdit`. Для даты рождения используется `QDateEdit` с всплывающим календарем. Поле телефонов принимает несколько номеров, разделенных запятыми.

```
ContactDialog::ContactDialog(QWidget *parent) : QDialog(parent){
    setWindowTitle("                    /
                    ");
    setMinimumWidth(400);

    QFormLayout *layout = new QFormLayout(this);

    lastNameEdit = new QLineEdit(this);
    firstNameEdit = new QLineEdit(this);
    middleNameEdit = new QLineEdit(this);
    addressEdit = new QLineEdit(this);
    birthDateEdit = new QDateEdit(this);
    birthDateEdit->setCalendarPopup(true);
    birthDateEdit->setDate(QDate::currentDate().addYears(-20));
    birthDateEdit->setMaximumDate(QDate::currentDate().addDays
       (-1));

    emailEdit = new QLineEdit(this);
    phonesEdit = new QLineEdit(this);
    phonesEdit->setPlaceholderText("
                                                ");

    layout->addRow("               :", lastNameEdit);
    layout->addRow("      :", firstNameEdit);
    layout->addRow("                 :", middleNameEdit);
    layout->addRow("          :", addressEdit);
    layout->addRow("                              :", birthDateEdit);
    layout->addRow("Email:", emailEdit);
    layout->addRow("                 :", phonesEdit);

    QHBoxLayout *buttonLayout = new QHBoxLayout();
    QPushButton *okButton = new QPushButton("OK", this);
    QPushButton *cancelButton = new QPushButton("             ",
       this);

    buttonLayout->addWidget(okButton);
    buttonLayout->addWidget(cancelButton);
    layout->addRow(buttonLayout);

    connect(okButton, &QPushButton::clicked, this, &ContactDialog
       ::validateAndAccept);
    connect(cancelButton, &QPushButton::clicked, this, &QDialog::
       reject);
}
```

Листинг 2: Конструктор ContactDialog

3.2.3 Класс PhoneBook

Класс PhoneBook представляет главное окно приложения и наследуется от QWidget. Класс управляет списком контактов и обеспечивает их отображение в таблице.

```
    class PhoneBook : public QWidget{
    Q_OBJECT

private:
    QTableWidget *table;
    QLineEdit *searchEdit;
    QList<Contact> contacts;
    QString filename;

    void updateTable();

private slots:
    void addContact();
    void editContact();
    void deleteContact();
    void searchByColumn(int column);
    void saveToFile();
    void loadFromFile();

public:
    PhoneBook(QWidget *parent = nullptr);
};
```

Листинг 3: Класс PhoneBook

Приватные поля:

- QTableWidget *table — таблица для отображения контактов;
- QLineEdit *searchEdit — поле ввода поискового запроса;
- QList<Contact> contacts — список всех контактов;
- QString filename — имя файла для сохранения данных.

Приватные методы:

void updateTable() — обновляет отображение таблицы после изменений в списке контактов. Временно отключает сортировку через setSortingEnabled(false), устанавливает количество строк равным размеру списка, заполняет каждую ячейку данными контакта, сохраняет индекс контакта в UserRole первой ячейки строки, включает сортировку обратно.

```
void PhoneBook::updateTable(){
    table->setSortingEnabled(false);
    table->setRowCount(contacts.size());

    for (int i = 0; i < contacts.size(); i++){
        const Contact &c = contacts[i];
        QTableWidgetItem *lastNameItem = new QTableWidgetItem(c.
           getLastName());
        lastNameItem->setData(Qt::UserRole, i);
        table->setItem(i, 0, lastNameItem);

        table->setItem(i, 1, new QTableWidgetItem(c.getFirstName
           ()));
        table->setItem(i, 2, new QTableWidgetItem(c.getMiddleName
           ()));
        table->setItem(i, 3, new QTableWidgetItem(c.getAddress())
           );
        table->setItem(i, 4, new QTableWidgetItem(c.getBirthDate
           ().toString("dd.MM.yyyy")));
        table->setItem(i, 5, new QTableWidgetItem(c.getEmail()));
        table->setItem(i, 6, new QTableWidgetItem(c.getPhones().
           join(", ")));
    }
    table->setSortingEnabled(true);
}
```

Листинг 4: Метод обновления таблицы

Приватные слоты:

void addContact() — создает экземпляр диалога ContactDialog, открывает его, при подтверждении получает данные через getContact(), добавляет контакт в список и обновляет таблицу.

```
void PhoneBook::addContact(){
    ContactDialog dialog(this);
    if (dialog.exec() == QDialog::Accepted){
        Contact contact = dialog.getContact();
        contacts.append(contact);
        updateTable();
    }
}
```

Листинг 5: Метод добавления контакта

void editContact() — получает номер выбранной строки, проверяет наличие выбора, извлекает реальный индекс контакта из UserRole (учитываем то, что могла быть использована сортировка, и после этого нужно чтобы редактировался выбранный контакт), открывает диалог с предзаполненными данными через setContact(), при подтверждении обновляет контакт в списке.

```
void PhoneBook::editContact(){
    int row = table->currentRow();
    if (row < 0){
        QMessageBox::warning(this, "             ", "

                                          ");
        return;
    }

    int index = table->item(row, 0)->data(Qt::UserRole).toInt();
    ContactDialog dialog(this);
    dialog.setContact(contacts[index]);

    if (dialog.exec() == QDialog::Accepted){
        contacts[index] = dialog.getContact();
        updateTable();
    }
}
```

Листинг 6: Метод редактирования контакта

void deleteContact() — получает номер выбранной строки, запрашивает подтверждение через QMessageBox::question, при подтверждении извлекает индекс из UserRole (учитываем то, что могла быть использована сортировка и нужно чтобы при удалении удалялся именно выбранный контакт), удаляет контакт через removeAt() и обновляет таблицу.

```
void PhoneBook::deleteContact(){
    int row = table->currentRow();
    if (row < 0){
        QMessageBox::warning(this, "             ", "

                             ");
        return;
    }

    QMessageBox::StandardButton reply;
    reply = QMessageBox::question(this, "
                                  ",
                                   "

                                                ?",
                                  QMessageBox::Yes | QMessageBox
                                     ::No);

    if (reply == QMessageBox::Yes){
        int index = table->item(row, 0)->data(Qt::UserRole).toInt
           ();
        contacts.removeAt(index);
        updateTable();
```

```
    }
}
```

Листинг 7: Метод удаления контакта

void searchByColumn(int column) — получает текст запроса, преобразует в нижний регистр, для каждой строки таблицы проверяет наличие текста в указанном столбце или во всех столбцах, для поля телефонов разделяет список и проверяет каждый номер отдельно, скрывает или показывает строку через setRowHidden().

```
void PhoneBook::searchByColumn(int column){
    QString text = searchEdit->text().trimmed().toLower();

    if(text.isEmpty()){
        for(int i=0; i<table->rowCount(); i++)
            table->setRowHidden(i, false);
        return;
    }

    for(int i=0; i<table->rowCount(); i++){
        bool found = false;

        if(column >= 0 && column < 6){
            if(table->item(i, column) && table->item(i, column)->
               text().toLower().contains(text)){
                found = true;
            }
        } else {
            if(table->item(i, 6)){
                QString phones = table->item(i, 6)->text().
                   toLower();
                QStringList phoneList = phones.split(",", Qt::
                   SkipEmptyParts);

                for(const QString &phone : phoneList){
                    if(phone.trimmed().contains(text)){
                        found = true;
                        break;
                    }
                }
            }
        }
        table->setRowHidden(i, !found);
    }
}
```

Листинг 8: Метод поиска по выбранному столбцу

void saveToFile() — открывает файл в режиме записи, создает QTextStream с кодировкой UTF-8, для каждого контакта записывает строковое представ-

ление через toString(), закрывает файл и выводит сообщение об успехе.

```
void PhoneBook::saveToFile(){
    QFile file(filename);
    if (!file.open(QIODevice::WriteOnly | QIODevice::Text)){
        QMessageBox::warning(this, "             ", "

                    ");
        return;
    }

    QTextStream out(&file);
    out.setEncoding(QStringConverter::Utf8);

    for (const Contact &c : contacts){
        out << c.toString() << "\n";
    }

    file.close();
    QMessageBox::information(this, "          ", "
                        ");
}
```

Листинг 9: Метод сохранения в файл

void loadFromFile() — открывает файл в режиме чтения, очищает текущий список, создает QTextStream с кодировкой UTF-8, читает строки до конца файла, для каждой строки создает контакт через fromString(), проверяет корректность парсинга, добавляет контакт в список и обновляет таблицу.

```
void PhoneBook::loadFromFile(){
    QFile file(filename);
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)){
        return;
    }

    contacts.clear();
    QTextStream in(&file);
    in.setEncoding(QStringConverter::Utf8);

    while (!in.atEnd()){
        QString line = in.readLine();
        Contact contact = Contact::fromString(line);
        if (!contact.getLastName().isEmpty()){
            contacts.append(contact);
        }
    }

    file.close();
    updateTable();
}
```

Листинг 10: Метод загрузки из файла

Конструктор PhoneBook(QWidget *parent):

Конструктор инициализирует главное окно приложения. Устанавливает путь к файлу как «домашняя_директория/phonebook.txt» через QDir::homePath(). Устанавливает заголовок окна и минимальный размер 800×600 пикселей.

Создает вертикальный layout. Формирует панель поиска с меткой, полем ввода и выпадающим списком для выбора поля. Заполняет QComboBox семью полями с соответствующими индексами от 0 до 6.

Создает таблицу QTableWidget с семью столбцами. Устанавливает заголовки столбцов. Включает автоматическую сортировку и растягивание последнего столбца.

Создает панель кнопок с пятью кнопками управления.

```
PhoneBook::PhoneBook(QWidget *parent) : QWidget(parent){
    filename = QDir::homePath() + "/phonebook.txt";

    setWindowTitle("                                          ");
    setMinimumSize(800, 600);

    QVBoxLayout *mainLayout = new QVBoxLayout(this);

    QHBoxLayout *searchLayout = new QHBoxLayout();
    QLabel *searchLabel = new QLabel("          :", this);
    searchEdit = new QLineEdit(this);
    searchLayout->addWidget(searchLabel);
    searchLayout->addWidget(searchEdit);
    mainLayout->addLayout(searchLayout);

    QComboBox *fieldCombo = new QComboBox(this);
    fieldCombo->addItem("              ", 0);
    fieldCombo->addItem("      ", 1);
    fieldCombo->addItem("                ", 2);
    fieldCombo->addItem("          ", 3);
    fieldCombo->addItem("                          ", 4);
    fieldCombo->addItem("Email", 5);
    fieldCombo->addItem("                ", 6);

    searchLayout->addWidget(searchLabel);
    searchLayout->addWidget(searchEdit);
    searchLayout->addWidget(fieldCombo);
    mainLayout->addLayout(searchLayout);

    connect(searchEdit, &QLineEdit::textChanged, this, [this,
       fieldCombo](){
    int column = fieldCombo->currentData().toInt();
        searchByColumn(column);
```

```
});

    table = new QTableWidget(this);
    table->setSelectionMode(QAbstractItemView::SingleSelection);
    table->setColumnCount(7);
    QStringList headers;
    headers << "               " << "       " << "                  "
        << "           "
            << "                              " << "Email" << "
                              ";
    table->setHorizontalHeaderLabels(headers);
    table->horizontalHeader()->setStretchLastSection(true);
    table->setSelectionBehavior(QAbstractItemView::SelectRows);
    table->setSortingEnabled(true);
    mainLayout->addWidget(table);

    QHBoxLayout *buttonLayout = new QHBoxLayout();
    QPushButton *addButton = new QPushButton("                ",
       this);
    QPushButton *editButton = new QPushButton("
                                  ", this);
    QPushButton *deleteButton = new QPushButton("               ",
        this);
    QPushButton *saveButton = new QPushButton("
       ", this);
    QPushButton *loadButton = new QPushButton("
       ", this);

    buttonLayout->addWidget(addButton);
    buttonLayout->addWidget(editButton);
    buttonLayout->addWidget(deleteButton);
    buttonLayout->addWidget(saveButton);
    buttonLayout->addWidget(loadButton);
    mainLayout->addLayout(buttonLayout);

    connect(addButton, &QPushButton::clicked, this, &PhoneBook::
       addContact);
    connect(editButton, &QPushButton::clicked, this, &PhoneBook::
       editContact);
    connect(deleteButton, &QPushButton::clicked, this, &PhoneBook
       ::deleteContact);
    connect(saveButton, &QPushButton::clicked, this, &PhoneBook::
       saveToFile);
    connect(loadButton, &QPushButton::clicked, this, &PhoneBook::
       loadFromFile);

    loadFromFile();
}
```

Листинг 11: Конструктор класса PhoneBook

Механизм сигналов и слотов:

В конструкторе PhoneBook все сигналы пользовательского интерфейса связываются с соответствующими слотами через функцию connect. Нажатие кнопки «Добавить» связано со слотом addContact(), кнопки «Редактировать» — с editContact(), кнопки «Удаление» — с deleteContact(). Кнопки сохранения и загрузки связаны с соответствующими слотами saveToFile() и loadFromFile(). Изменение текста в поле поиска автоматически вызывает слот searchByColumn() через лямбда-функцию, которая получает индекс выбранного поля из выпадающего списка.

3.3 Валидация данных

Валидация данных реализована в отдельных методах.

3.3.1 Валидация имени, фамилии и отчества контакта

Метод `validateName` выполняет комплексную проверку строки, представляющей имя, фамилию или отчество контакта. Метод принимает два параметра: саму строку для проверки и название поля (например, "lastName"), которое используется в сообщениях об ошибках.

Логика валидации включает следующие проверки: проверка на пустоту – удаление лишних пробелов и проверка на пустую строку, проверка первой буквы – первый символ должен быть заглавной буквой, проверка позиций дефиса – строка не должна начинаться или заканчиваться дефисом, проверка допустимых символов – разрешены буквы, цифры, пробелы и дефисы, проверка регистра – формат "Первая заглавная, остальные строчные".

Особенности регулярного выражения `nameRegex`:

Регулярное выражение `"^[\\p{L}\\d\\s-]+$"` содержит следующие компоненты:

- `^` — начало строки (anchor start);
- `[\\p{L}\\d\\s-]` — символьный класс, разрешающий:
  1. `\\p{L}` — любые буквы из всех языков;
  2. `\\d` — цифры (0–9);
  3. `\\s` — пробельные символы (пробел, табуляция, перенос строки и др.);
  4. `-` — дефис.
- `+` — один или более символов из указанного класса (квантификатор);
- `$` — конец строки (anchor end).

Это выражение гарантирует, что строка содержит только допустимые символы и не имеет нежелательных символов в начале или конце. Все проверки выполняются последовательно, и при обнаружении первой ошибки метод возвращает `false` с выводом соответствующего сообщения через `QMessageBox::warnin`

```
bool ContactDialog::validateName(const QString &name, const
   QString &fieldName){
    QString trimmed = name.trimmed();

    if (trimmed.isEmpty()){
        QMessageBox::warning(this, "            ", fieldName + "
                                                ");
        return false;
    }

    if (!trimmed[0].isUpper()){
        QMessageBox::warning(this, "            ", fieldName + "

                                           ");
        return false;
    }

    if (trimmed.startsWith('-') || trimmed.endsWith('-')){
        QMessageBox::warning(this, "            ", fieldName + "

                                                ");
        return false;
    }

    QRegularExpression nameRegex("^[\\p{L}\\d\\s-]+$");
    if (!nameRegex.match(trimmed).hasMatch()){
        QMessageBox::warning(this, "            ", fieldName + "
                                                                ,
                    ,                             ");
        return false;
    }

    QString normalized = trimmed.left(1).toUpper() + trimmed.mid
       (1).toLower(); //
                        ,                                       (2
                    )
    if (normalized != trimmed){
        QMessageBox::warning(this, "            ", fieldName + "
                                                  :

                                ,
                               ");
        return false;
    }
    return true;
}
```

Листинг 12: Валидация имени

3.3.2 Валидация адреса электронной почты

Метод `validateEmail` выполняет проверку строки, представляющей адрес электронной почты. Метод принимает один параметр — строку для проверки. Логика валидации включает следующие шаги: удаление пробелов в начале и конце строки, удаление всех внутренних пробелов, проверка на пустую строку, проверка соответствия регулярному выражению, описывающему структуру адреса электронной почты. В случае несоответствия выводится сообщение об ошибке через `QMessageBox::warning`.

Особенности регулярного выражения `emailRegex`:

Регулярное выражение `"^[a-zA-Z0-9]+@[a-zA-Z0-9]+\\.[a-zA-Z0-9]+$"` содержит следующие компоненты:

- `^` — начало строки (anchor start);
- `[a-zA-Z0-9]+` — одна или более латинских букв и цифр для локальной части адреса;
- `@` — обязательный символ «собака», разделяющий локальную часть и домен;
- `[a-zA-Z0-9]+` — одна или более латинских букв и цифр для имени домена;
- `\\.` — обязательная точка, разделяющая домен и зону;
- `[a-zA-Z0-9]+` — одна или более латинских букв и цифр для доменной зоны (например, «com», «ru»);
- `$` — конец строки (anchor end).

Это выражение гарантирует, что адрес электронной почты имеет базовую структуру вида `имя@домен.зона`, где каждая часть состоит только из латинских букв и цифр. Все проверки выполняются последовательно, и при обнаружении первой ошибки метод возвращает `false` с выводом соответствующего сообщения через `QMessageBox::warning`.

```
1 bool ContactDialog::validateEmail(const QString &email){
2     QString trimmed = email.trimmed();
3     trimmed.remove(' ');
```

```
4
5       if (trimmed.isEmpty()){
6           QMessageBox::warning(this, "                ", "Email
                                                    ");
7           return false;
8       }
9
10      QRegularExpression emailRegex("^[a-zA-Z0-9]+@[a-zA-Z0-9]+\\.[
            a-zA-Z0-9]+$");
11      if (!emailRegex.match(trimmed).hasMatch()){
12          QMessageBox::warning(this, "                ", "
                                                email");
13          return false;
14      }
15      return true;
16  }
```

Листинг 13: Валидация email

3.3.3 Валидация номера телефона

Метод `validatePhones` выполняет проверку строки, представляющей один или несколько телефонных номеров. Метод принимает один параметр — строку для проверки. Логика валидации включает следующие шаги: проверка на пустую строку, проверка отсутствия скобок, разделение строки на список номеров по запятой, нормализация каждого номера до последовательности цифр и символа «+», проверка минимальной длины нормализованного номера. При обнаружении ошибки выводится сообщение через `QMessageBox::warning`.

Особенности вспомогательной функции `normalizePhone`:

Функция `normalizePhone` проходит по каждому символу исходной строки и формирует новую строку, содержащую только цифры и символ «+». Это гарантирует, что дальнейшая проверка выполняется по унифицированному формату, исключающему пробелы, дефисы и другие символы.

```
1   QString ContactDialog::normalizePhone(const QString &phone) const
        {
2       QString result;
3       for (QChar c : phone){
4           if (c.isDigit() || c == '+'){
5               result += c;
6           }
7       }
8       return result;
9   }
```

Листинг 14: Вспомогательная функция нормализация телефона

Особенности проверки в методе `validatePhones`:

Проверка включает:

- `phones.trimmed().isEmpty()` — строка не должна быть пустой;
- `phones.contains('(') | phones.contains(')')` — номера не должны содержать скобки;
- `phones.split(',')` — поддержка нескольких номеров, разделённых запятой;
- `normalizePhone(phone.trimmed())` — нормализация каждого номера;
- `normalized.length() < 10` — минимальная длина номера должна быть не менее 10 символов.

```
bool ContactDialog::validatePhones(const QString &phones){
    if (phones.trimmed().isEmpty()){
        QMessageBox::warning(this, "            ", "
                                    ");
        return false;
    }

    if (phones.contains('(') || phones.contains(')')){
        QMessageBox::warning(this, "              ", "
                                                ,
                                                    ");
        return false;
    }

    QStringList phoneList = phones.split(',');
    for (const QString &phone : phoneList){
        QString normalized = normalizePhone(phone.trimmed());
        if (normalized.length() < 10){
            QMessageBox::warning(this, "             ", "
                                                        : "
                  + phone);
            return false;
        }
    }
    return true;
}
```

Листинг 15: Валидация номера телефона

3.3.4 Валидация адреса

Метод `validateAddress` выполняет проверку строки, представляющей адрес контакта. Метод принимает один параметр — строку для проверки. Логика валидации включает следующие шаги: удаление пробелов в начале и конце строки, проверка на наличие запрещённого символа «|». В случае обнаружения символа выводится сообщение об ошибке через `QMessageBox::warning`, и метод возвращает `false`. Если ошибок не найдено, метод возвращает `true`.

```
bool ContactDialog::validateAddress(const QString &address){
    QString trimmed = address.trimmed();

    if (trimmed.contains('|')){
        QMessageBox::warning(this, "            ", "
                                                    '|'");
        return false;
    }
    return true;
}
```

Листинг 16: Валидация адреса

3.3.5 Комплексная проверка и подтверждение данных

Метод `validateAndAccept` выполняет последовательную проверку всех полей формы контакта. Метод не принимает параметров, а обращается к значениям элементов интерфейса. Логика работы включает следующие шаги: проверка фамилии через `validateName`, проверка имени через `validateName`, проверка отчества (если поле не пустое) через `validateName`, проверка адреса через `validateAddress`, проверка электронной почты через `validateEmail`, проверка телефонов через `validatePhones`.

Если хотя бы одна проверка возвращает `false`, метод завершает выполнение без сохранения данных. Если все проверки пройдены успешно, вызывается метод `accept()`, который закрывает диалоговое окно и подтверждает ввод данных.

```
void ContactDialog::validateAndAccept(){
    if (!validateName(lastNameEdit->text(), "              "))
       return;
    if (!validateName(firstNameEdit->text(), "       ")) return;
    if (!middleNameEdit->text().trimmed().isEmpty() && !
       validateName(middleNameEdit->text(), "                 "))
       return;
    if (!validateAddress(addressEdit->text())) return;
    if (!validateEmail(emailEdit->text())) return;
    if (!validatePhones(phonesEdit->text())) return;

```

```
9       accept();
10  }
```

Листинг 17: Комплексная проверка и подтверждение

4 Тестирование приложения

4.1 Начальное состояние приложения

При первом запуске приложения отображается главное окно с пустой таблицей контактов (см. рисунок ??), полем поиска в верхней части и кнопками управления в нижней части.

Главное окно имеет минимальный размер 800x600 пикселей. Таблица настроена на автоматическое растягивание последнего столбца для заполнения всей ширины окна. Режим выбора установлен на SelectRows, что позволяет выделять только целые строки.

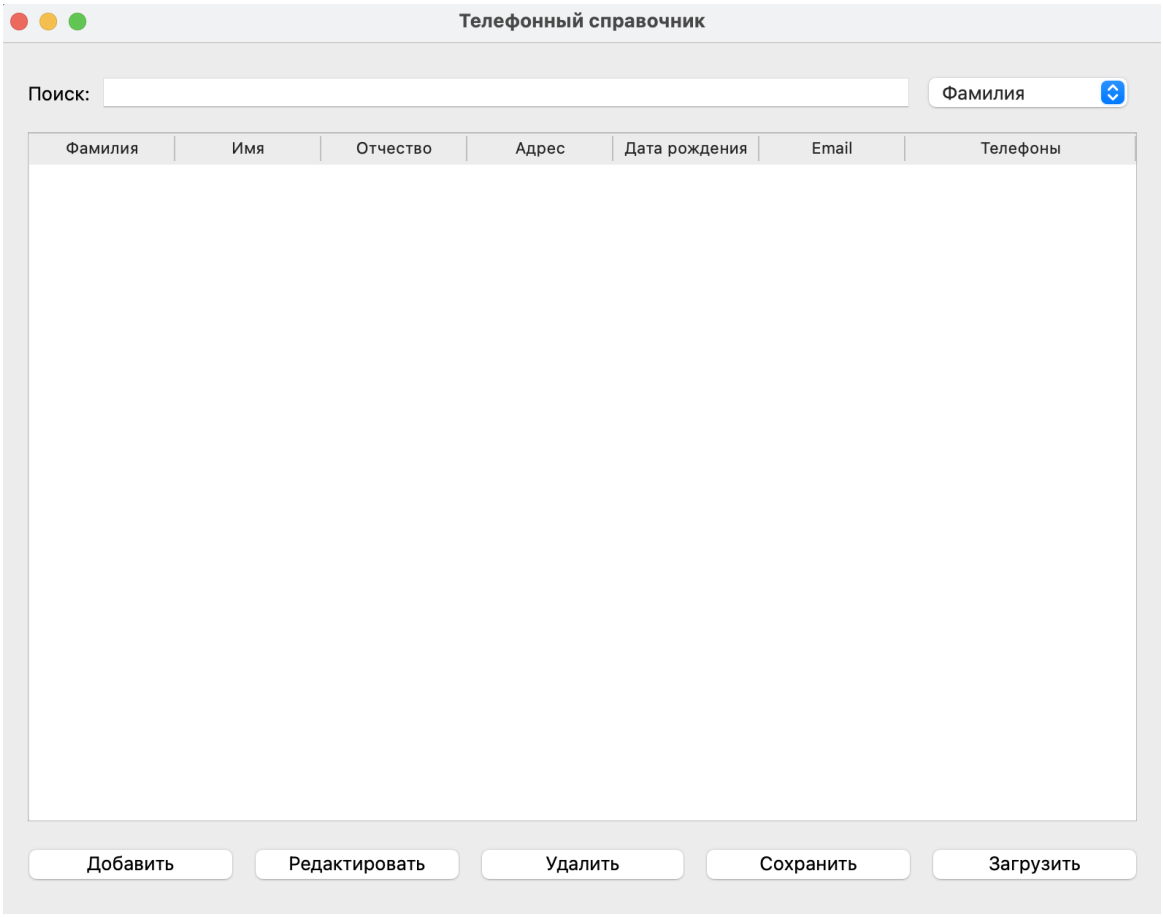

Рис. 1: Главное окно при первом запуске

4.2 Добавление нового контакта

Пользователь нажимает кнопку «Добавить», после чего открывается диалоговое окно с формой ввода (см. рисунок ??). Пользователь заполняет все обязательные поля: фамилию «Иванов», имя «Иван», отчество «Иванович», адрес «Москва, ул. Ленина, д. 1», дату рождения через всплывающий календарь, email
«ivanov@example.com» и телефоны «+79991234567, +74951234567».

После нажатия кнопки OK система выполняет валидацию всех полей. Если все данные корректны, диалоговое окно закрывается, новый контакт добавляется в список, и таблица обновляется. Пользователь видит новую строку с введенными данными. Данные автоматически сохраняются в файл phonebook.txt.

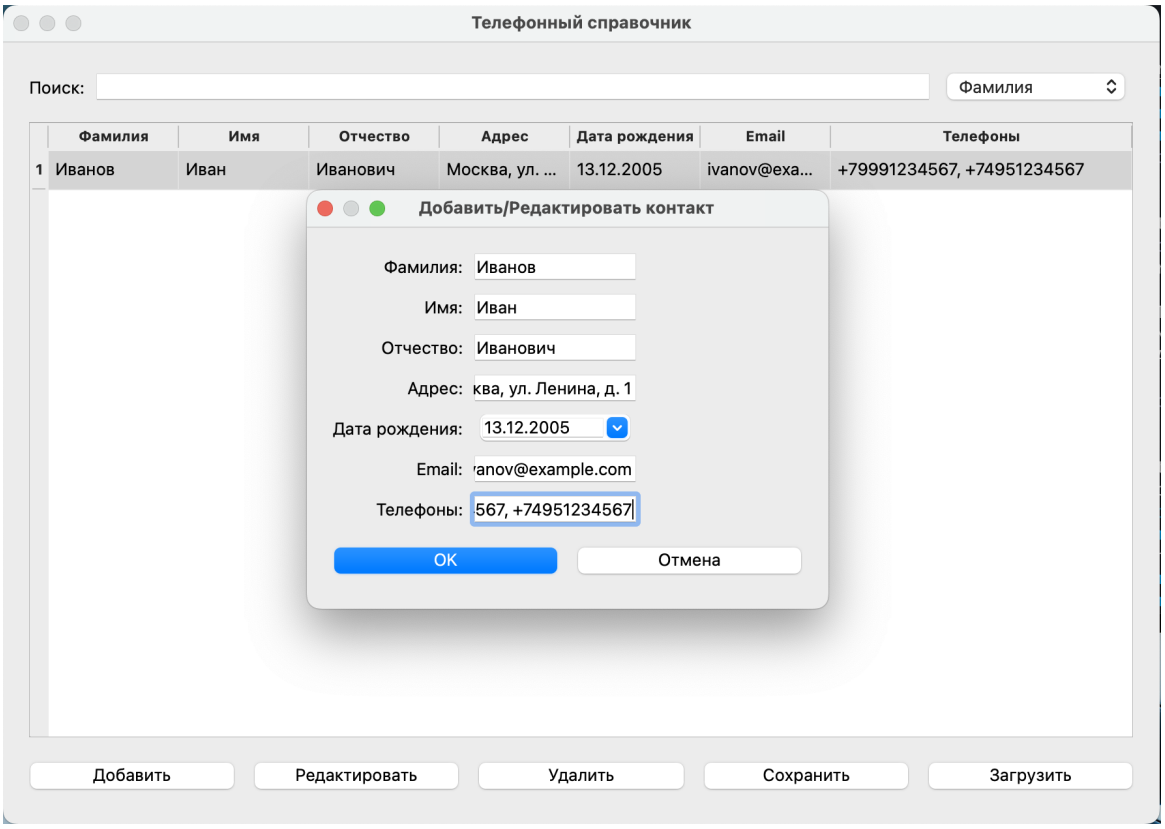

Рис. 2: Добавление контакта

4.3 Валидация некорректных данных

При попытке добавить контакт с некорректными данными система выводит соответствующие сообщения об ошибках.

4.3.1 Некорректное имя

Если пользователь вводит имя с маленькой буквы, например «иван» (см. рисунок ??), система выводит сообщение «Имя должно начинаться с заглавной буквы». Диалоговое окно остается открытым, позволяя пользователю исправить ошибку.

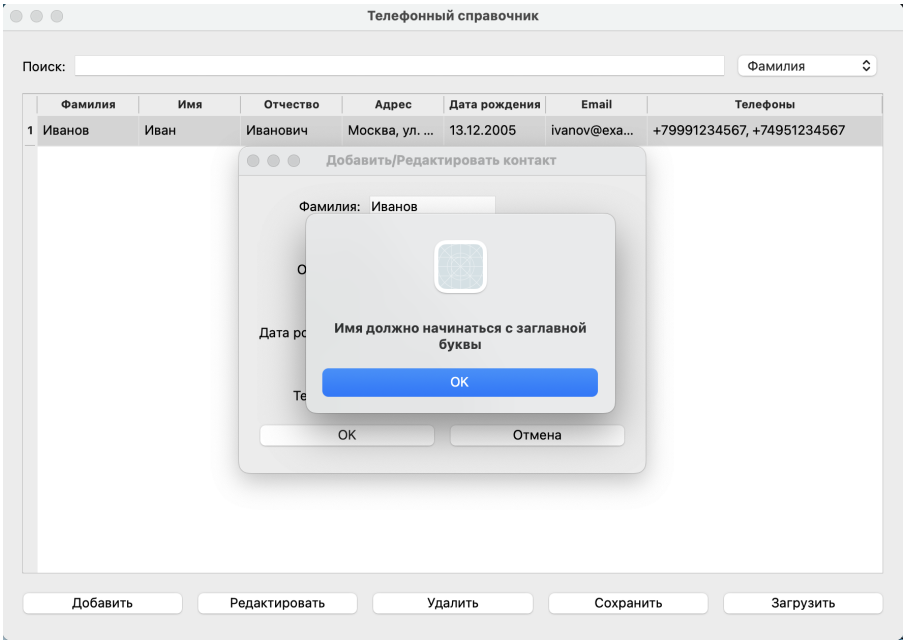

Рис. 3: Ввод некорректного имени

4.3.2 Некорректный email

Если пользователь вводит email без символа @ или без доменной зоны, например «ivanovexample.com», как на рисунке **??**, или «ivanov@example», система выводит сообщение «Неверный формат email». Пользователь должен исправить формат адреса перед сохранением.

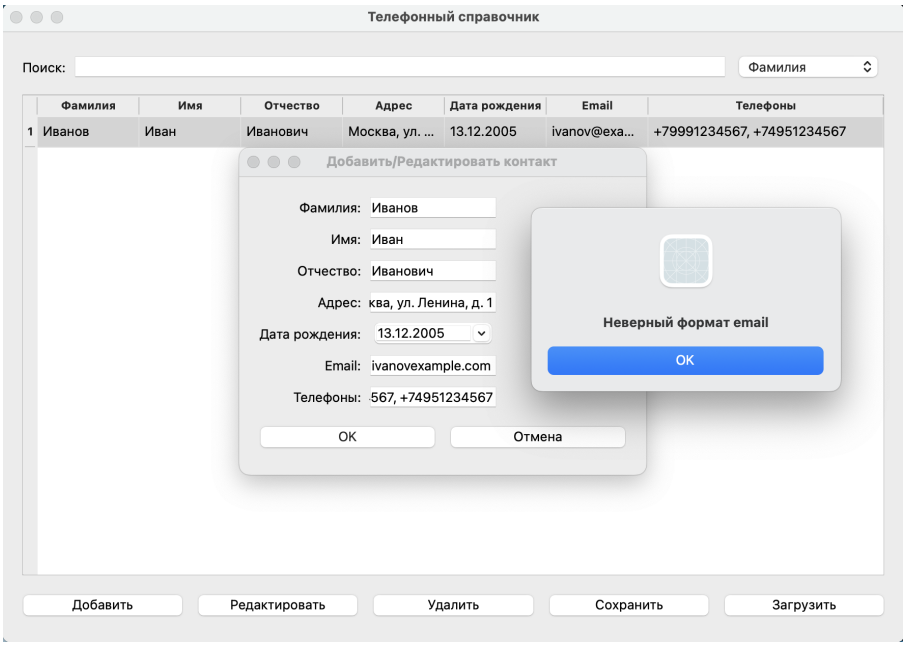

Рис. 4: Ввод некорректной почты

4.3.3 Некорректный телефон

Если пользователь вводит слишком короткий номер телефона, например «123» (см. рисунок ??), система выводит сообщение «Телефон слишком короткий: 123». Минимальная длина нормализованного номера должна составлять 10 символов.

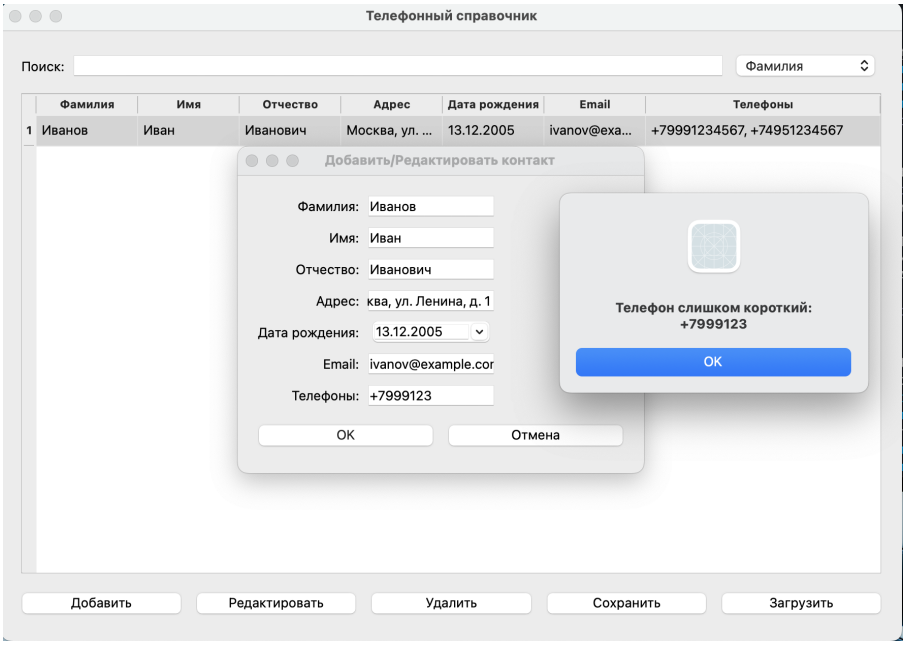

Рис. 5: Ввод некорректного мобильного телефона

Также учитывается случай, когда пользователь в номер телефона вводит символы, пусть латиницей. Они при записи просто перестают учитываться и не записываются в поле контакта (см. рисунок ??).

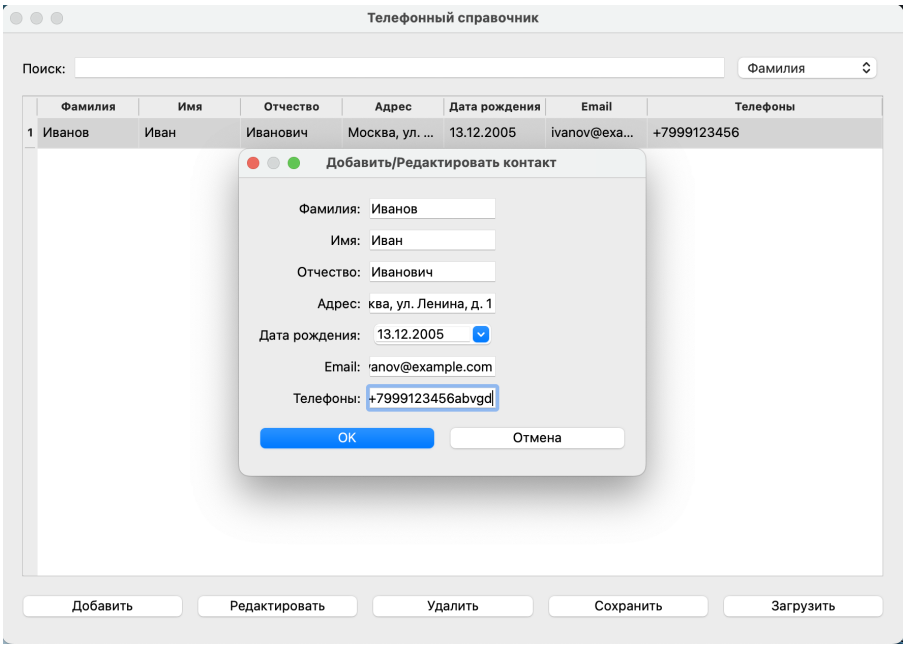

Рис. 6: Ввод символов в поле «Телефоны»

4.3.4 fieldName с дефисом в начале

Если пользователь вводит фамилию (имя, отчество), начинающуюся с дефиса, например «-Иванов», система выводит сообщение «Фамилия должна начинаться с заглавной буквы».

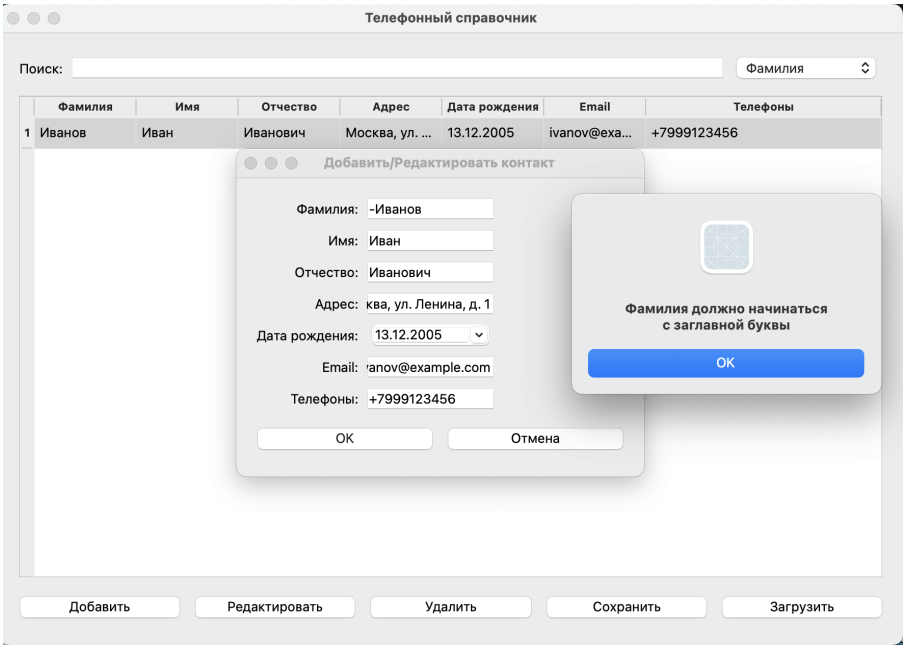

Рис. 7: Пример некорректного заполнения поля «Фамилия»

4.4 Редактирование существующего контакта

Пользователь выбирает строку в таблице, кликая по ней. После выбора становится активной кнопка «Редактировать». При нажатии на эту кнопку открывается диалоговое окно ContactDialog, все поля которого уже заполнены данными выбранного контакта.

Пользователь изменяет необходимые поля, например, обновляет адрес на «Санкт-Петербург, Невский проспект, д. 10» и добавляет еще один телефон. После нажатия кнопки OK система выполняет валидацию измененных данных. Если все данные корректны, обновленный контакт заменяет старую запись в списке, таблица обновляется, показывая новые данные. Все действия показаны на скриншотах ?? и ??.

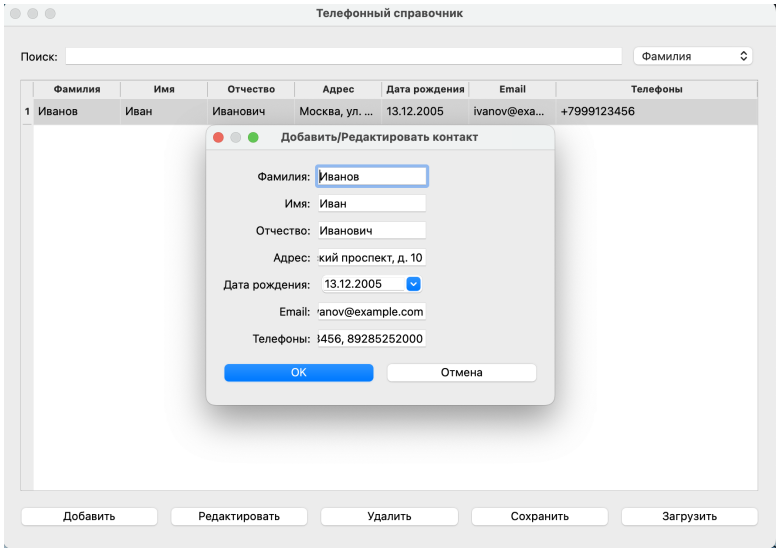

Рис. 8: Процесс редактирования

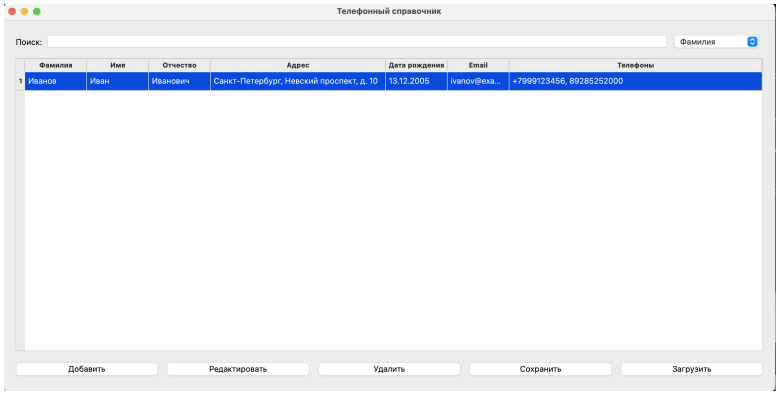

Рис. 9: Результат редактирования

4.5 Удаление контакта

Пользователь выбирает строку в таблице, после чего становится активной кнопка «Удалить». При нажатии на кнопку система выводит диалоговое окно с вопросом «Удалить выбранный контакт?» и кнопками «Yes» и «No», как показано на рисунке **??**. Также программа учитывает, что всегда выделен какой-либо контакт пользователем, т.е. множественное удаление не может происходить - так как отключено множественное выделение:

```
table->setSelectionMode(QAbstractItemView::SingleSelection);
```

Если пользователь подтверждает удаление, нажимая «Yes», контакт удаляется из списка contacts, строка исчезает из таблицы. Если пользователь нажимает «No», операция отменяется, и контакт остается в справочнике.

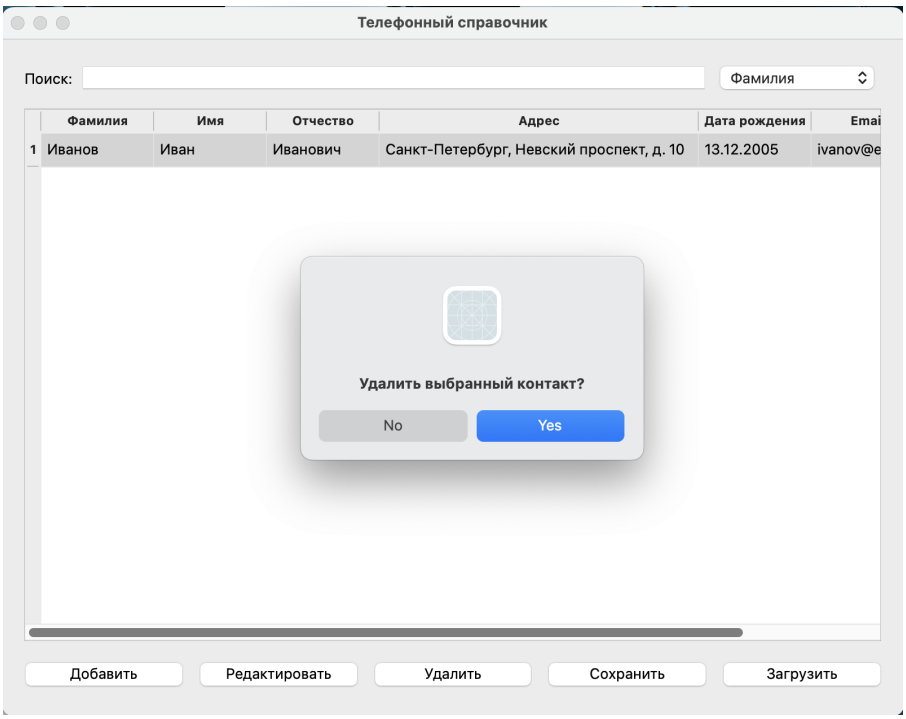

Рис. 10: Удаление выбранного контакта

4.6 Поиск контактов

4.6.1 Поиск по фамилии

Пользователь вводит текст «Иванов» в поле поиска. Система в реальном времени фильтрует таблицу, оставляя видимыми только те строки, которые содержат введенную подстроку в любом из полей. Все остальные строки скрываются.

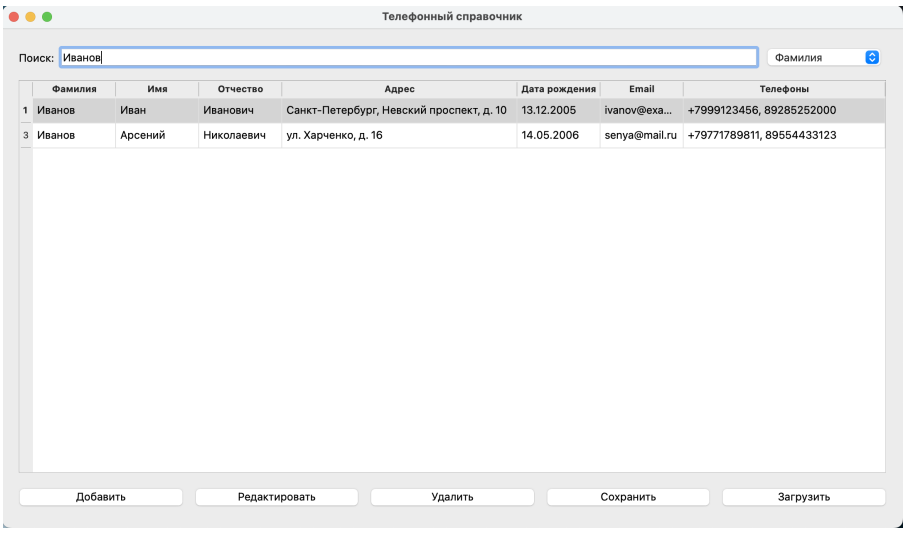

Рис. 11: Поиск по фамилии

4.6.2 Поиск по email

Пользователь вводит часть email, например «@example». Система показывает только те контакты, у которых в поле email присутствует данная подстрока. Поиск не чувствителен к регистру символов.

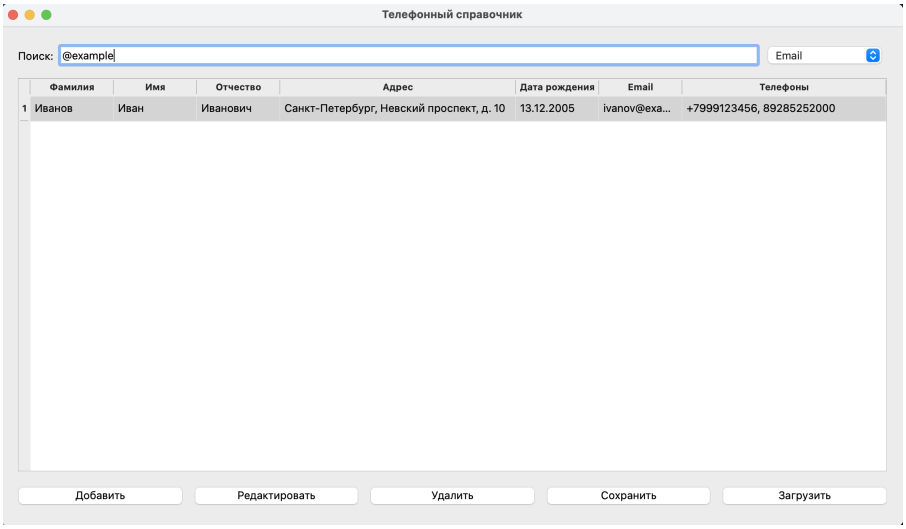

Рис. 12: Поиск по почте

4.6.3 Поиск по телефону

Пользователь вводит часть телефонного номера, например «999». Система показывает все контакты, у которых хотя бы один телефон содержит эту последовательность цифр.

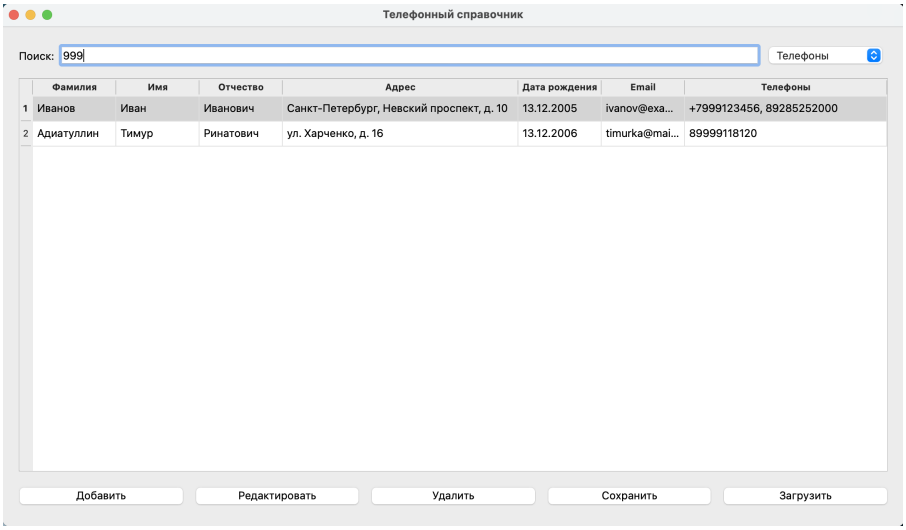

Рис. 13: Поиск по телефону

Рассмотрим ситуацию: когда у контакта два телефона, так как они записываются через запятую, то поиск не учитывает эту запятую, а рассматривает только цифры в номере (см. рисунок **??**). Одна из микроособенностей данной программы.

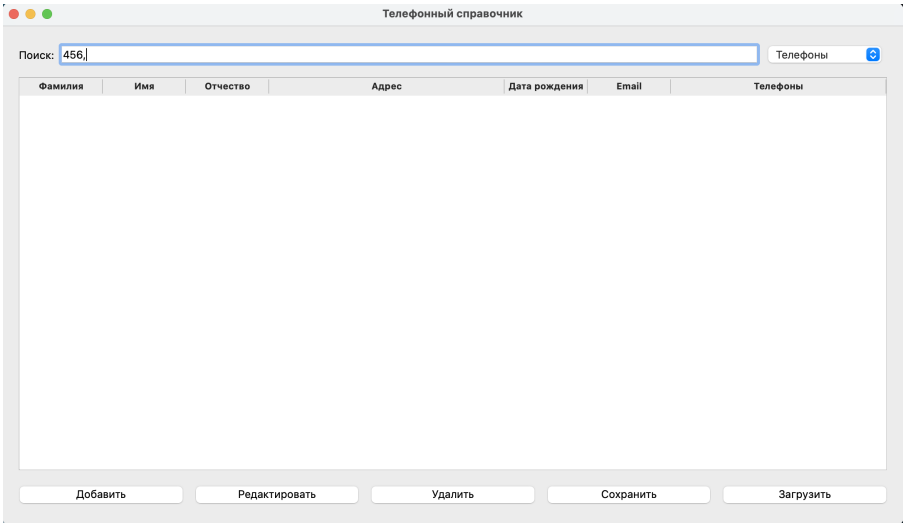

Рис. 14: Проверка на поиск при вводе сразу двух номеров

4.6.4 Очистка поиска

Когда пользователь очищает поле поиска, все строки таблицы снова становятся видимыми, показывая полный список контактов.

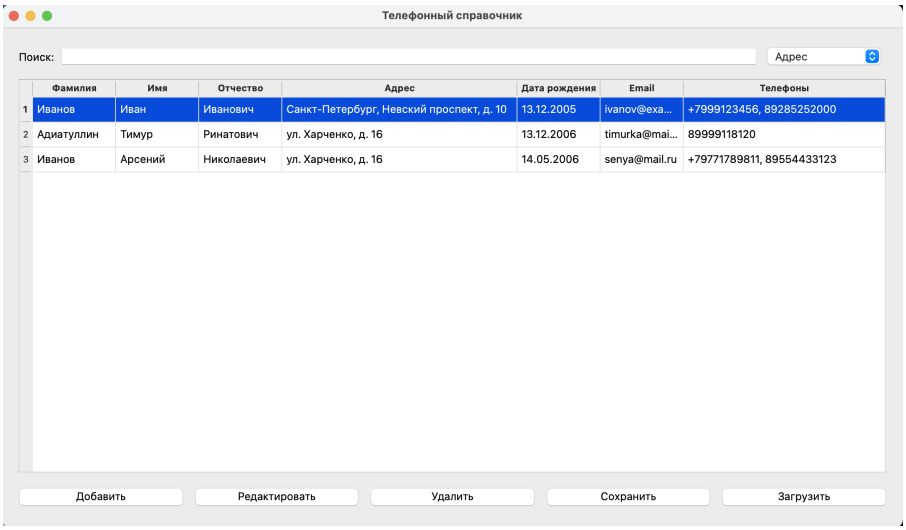

Рис. 15: Пустой поиск

4.7 Сортировка данных

4.7.1 Сортировка по фамилии

Пользователь кликает на заголовок столбца «Фамилия». Система автоматически сортирует все строки таблицы по фамилии в алфавитном порядке по возрастанию (см. рисунок **??**). При повторном клике на тот же заголовок направление сортировки меняется на убывание.

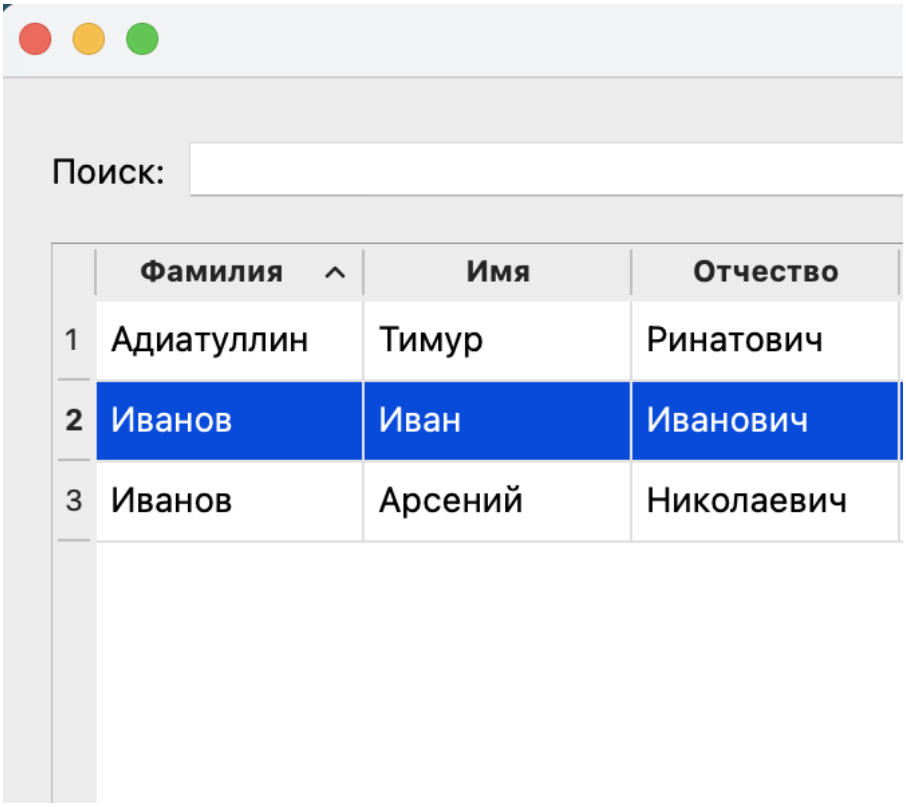

Рис. 16: Сортировка в алфавитном порядке

4.7.2 Сортировка по дате рождения

Пользователь кликает на заголовок столбца «Дата рождения». Система сортирует контакты по датам, располагая их от самых старых к самым молодым (см. рисунок ??). Повторный клик меняет порядок на обратный.

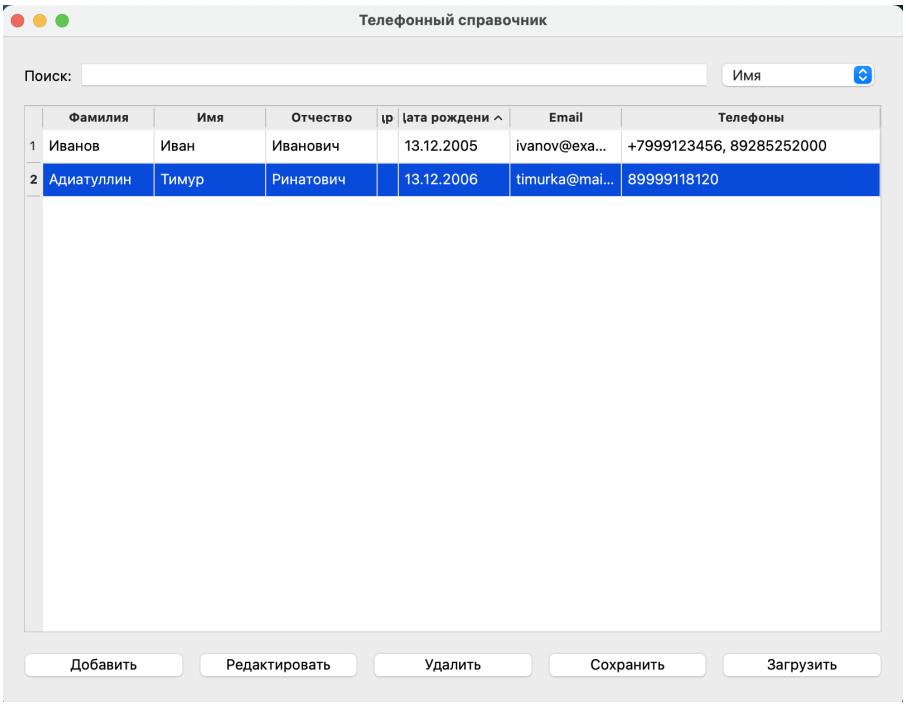

Рис. 17: Сортировка по дате

4.7.3 Сортировка по email

Система поддерживает сортировку по любому столбцу таблицы. Клик по заголовку столбца «Email» сортирует контакты в алфавитном порядке по адресам электронной почты (см. рисунок ??).

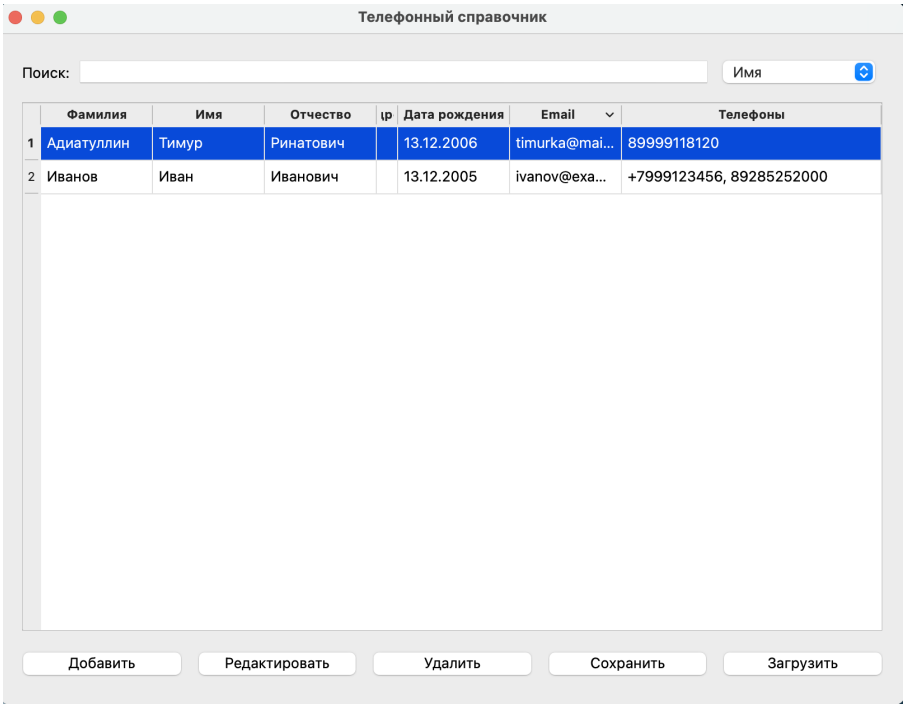

Рис. 18: сортировка в алфавитном порядке по полю «Email»

Механизм сортировки реализован встроенными средствами QTableWidget через флаг setSortingEnabled(true), что обеспечивает автоматическую обработку кликов по заголовкам столбцов.

4.8 Работа с файлами

4.8.1 Автоматическая загрузка при запуске

При запуске приложения конструктор PhoneBook вызывает метод loadFromFile, который пытается открыть файл phonebook.txt. Если файл существует и содержит корректные данные, все контакты загружаются в память и отображаются в таблице. Если файл не существует, приложение запускается с пустым справочником.

4.8.2 Автоматическое сохранение после изменений

После каждой операции добавления, редактирования или удаления контакта метод updateTable вызывает saveToFile, который записывает текущее состояние списка контактов в файл. Это гарантирует, что данные не будут потеряны при внезапном закрытии приложения.

4.8.3 Ручное сохранение

Пользователь может явно вызвать сохранение данных, нажав кнопку «Сохранить». Если операция выполнена успешно, система выводит сообщение «Данные сохранены» (см. рисунок ??). Если возникла ошибка при открытии файла, отображается предупреждение «Не удалось открыть файл для записи».

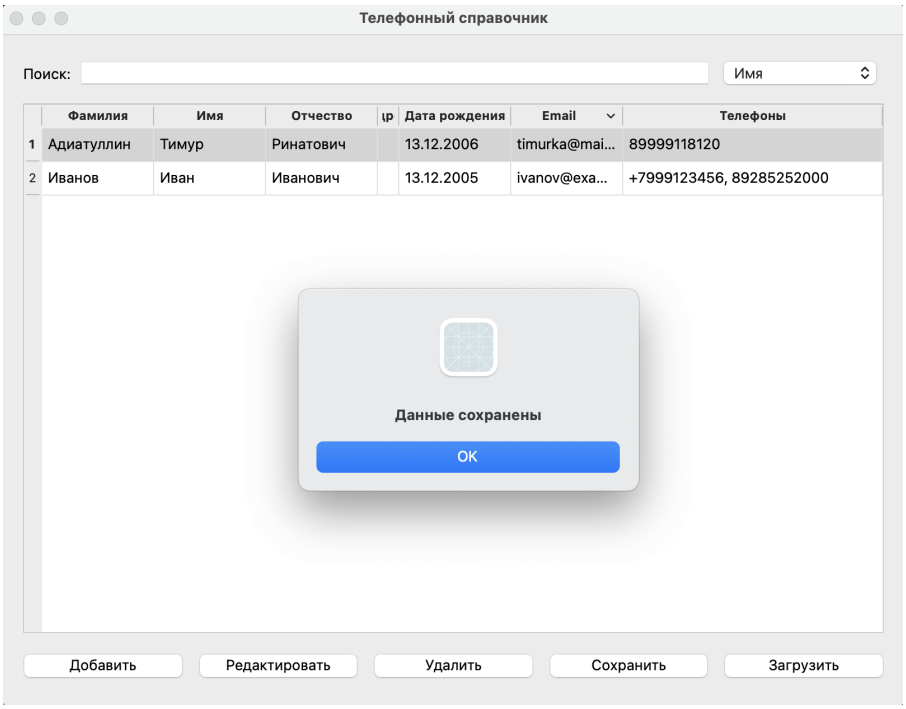

Рис. 19: Ручное сохранение в файл

4.8.4 Ручная загрузка

Кнопка «Загрузить» позволяет перезагрузить данные из файла. Это полезно, если файл был изменен внешним приложением. Текущее содержимое списка contacts полностью заменяется данными из файла, после чего таблица обновляется.

Сами же данные в файле записаны следующим образом:

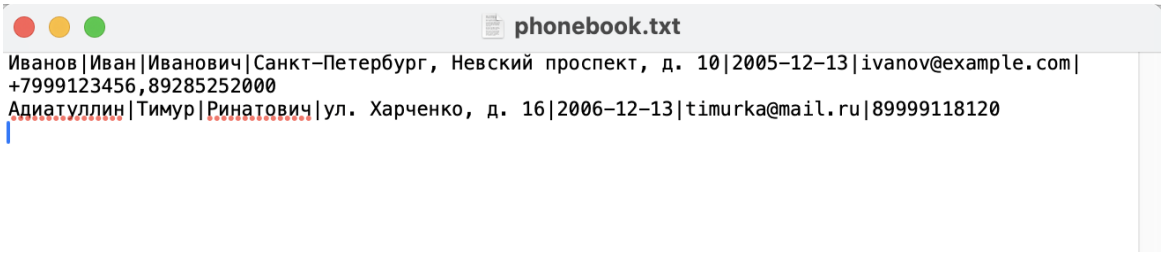

Рис. 20: phonebook.txt

4.9 Использование календаря для выбора даты

При добавлении или редактировании контакта пользователь может выбрать дату рождения с помощью виджета QDateEdit. Справа от поля даты располагается кнопка с иконкой календаря. При клике на эту кнопку появляется всплывающий календарь QCalendarWidget.

Пользователь может ориентироваться по месяцам и годам, используя стрелки в верхней части календаря. Клик по конкретной дате закрывает календарь и устанавливает выбранное значение в поле даты. Виджет настроен так, что максимально допустимая дата — это вчерашний день, предотвращая выбор будущих дат или текущего дня. Пример приведен на рисунке **??**.

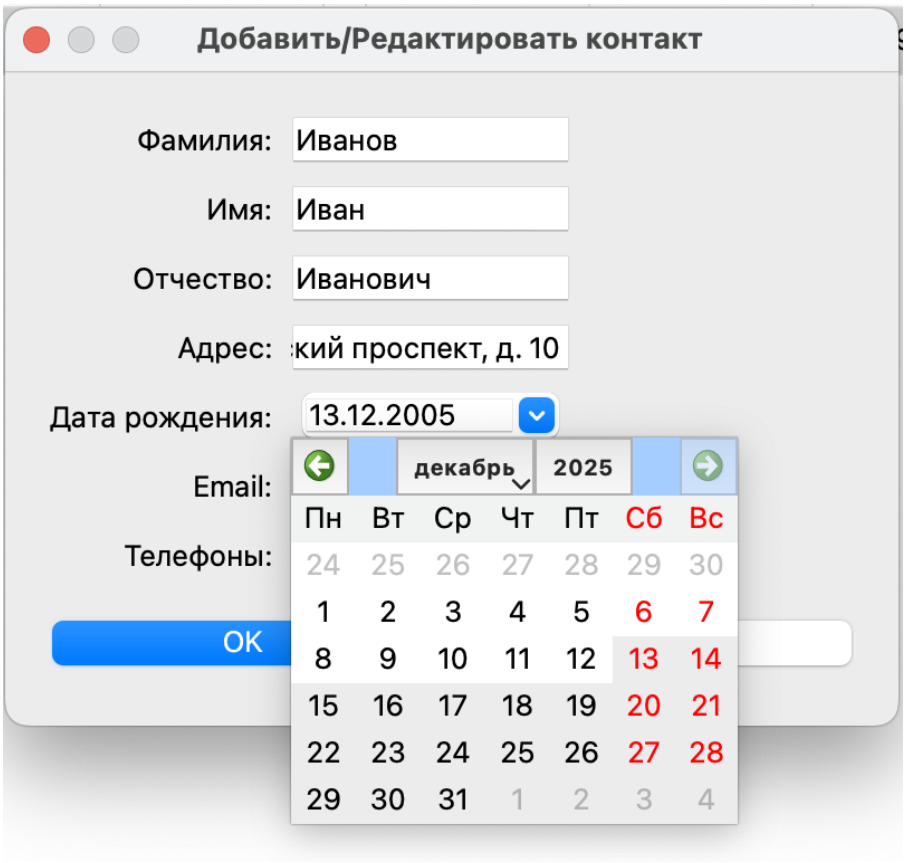

Рис. 21: Виджет-календарь

4.10 Тестирование регулярных выражений

Для проверки корректности работы валидации был проведен ряд тестов с различными входными значениями.

4.10.1 Тестирование валидации имени

Имя «Иван» проходит валидацию успешно. Имя «иван» отклоняется с сообщением о необходимости заглавной буквы. Имя «Иван-Петр» проходит валидацию, так как дефис допустим внутри строки. Имя «-Иван» отклоняется,

так как не может начинаться с дефиса. Имя «Иван123» проходит валидацию, поскольку цифры разрешены.

4.10.2 Тестирование валидации email

Адрес «test@mail.ru» проходит валидацию успешно. Адрес «test@mail» отклоняется из-за отсутствия доменной зоны. Адрес «testmail.ru» отклоняется из-за отсутствия символа @. Адрес «test @mail.ru» корректируется автоматически путем удаления всех пробелов перед проверкой.

4.10.3 Тестирование валидации телефона

Номер «+79991234567» проходит валидацию и нормализуется до «+79991234567». Номер «8(999)123-45-67» проходит валидацию и нормализуется до «89991234567». Номер «123» отклоняется как слишком короткий. Номер «+7 999 123 45 67» проходит валидацию и нормализуется до «+79991234567».

4.10.4 Тестирование валидации даты

Дата 15.03.2000 проходит валидацию успешно. Текущая дата или любая будущая дата отклоняется системой, так как виджет QDateEdit настроен с максимальным значением на вчерашний день.

4.11 Результаты тестирования

В результате тестирования были проверены следующие аспекты функциональности приложения.

4.11.1 Функциональность операций

Все операции добавления, чтения, обновления и удаления контактов работают корректно. Новые контакты успешно добавляются в список и отображаются в таблице. Редактирование позволяет изменять любые поля существующего контакта. Удаление корректно удаляет выбранный контакт после подтверждения пользователя.

4.11.2 Валидация данных

Система эффективно отсекает некорректные данные на этапе ввода. Пользователь получает понятные сообщения об ошибках с указанием конкретного

поля и характера проблемы. Регулярные выражения правильно проверяют формат имен, email и телефонных номеров.

4.11.3 Поиск и фильтрация

Система корректно фильтрует данные в реальном времени по мере ввода поискового запроса. Поиск работает по всем полям таблицы одновременно. Реализация через скрытие строк обеспечивает мгновенную реакцию на действия пользователя.

4.11.4 Сортировка

Механизм сортировки работает корректно для всех столбцов таблицы. Пользователь может сортировать данные как по текстовым полям, так и по дате. Смена направления сортировки при повторном клике работает ожидаемым образом.

4.11.5 Работа с файлами

Сохранение и загрузка данных происходят без потери информации. Формат хранения с использованием разделителей обеспечивает простоту и читаемость файла.

4.11.6 Интерфейс

Все элементы интерфейса корректно реагируют на действия пользователя. Состояние кнопок меняется в зависимости от контекста. Диалоговые окна правильно открываются и закрываются. Таблица адекватно отображает данные и поддерживает интерактивное взаимодействие.

5 Список использованных источников

1. Qt Documentation. Qt 6 Documentation. URL: https://doc.qt.io/qt-6/ (дата обращения: 13.11.2025).
2. Qt Documentation. Qt Widgets Module. URL: https://doc.qt.io/qt-6/qtwidgets-index.html (дата обращения: 14.11.2025).
3. Qt Documentation. Signals & Slots. URL: https://doc.qt.io/qt-6/signalsandslot
html (дата обращения: 14.11.2025).
4. Qt Documentation. QTableWidget Class. URL: https://doc.qt.io/qt-6/qtablewidget.html (дата обращения: 14.11.2025).
5. Qt Documentation. QDialog Class. URL: https://doc.qt.io/qt-6/qdialog.html (дата обращения: 15.11.2025).
6. Qt Documentation. QRegularExpression Class. URL: https://doc.qt.io/qt-6/qregularexpression.html (дата обращения: 15.11.2025).
7. Qt Documentation. QFile Class. URL: https://doc.qt.io/qt-6/qfile.html (дата обращения: 16.11.2025).
8. Qt Documentation. QDateEdit Class. URL: https://doc.qt.io/qt-6/qdatetimeedi
html (дата обращения: 16.11.2025).
9. cppreference.com. C++ Standard Library Reference. URL: https://en.cppreference
com/ (дата обращения: 16.11.2025).
10. Qt Documentation. Layout Management. URL: https://doc.qt.io/qt-6/layout.html (дата обращения: 16.11.2025).