Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Программирование на языках высокого уровня

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА к курсовому проекту на тему

АДРЕСНАЯ КНИГА С ПОДДЕРЖКОЙ ФОТОГРАФИЙ  
БГУИР КП 1-40 02 01 412 ПЗ

Студент: группы 150504, Лужков И.А.

Руководитель: Марзалюк А.В.

Минск 2022

**СОДЕРЖАНИЕ**

**ОГЛАВЛЕНИЕ**

[**ВВЕДЕНИЕ 3**](#_Toc122062436)

[**1. ОБЗОР ЛИТЕРАТУРЫ 5**](#_Toc122062437)

[**2.СТРУКТУРНОЕ ПРОЕКТРИРОВАНИЕ 12**](#_Toc122062438)

[**3. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 16**](#_Toc122062439)

[**4. РУКОВОДСТОВО ПОЛЬЗОВАТЕЛЯ 20**](#_Toc122062440)

[**ЗАКЛЮЧЕНИЕ 27**](#_Toc122062441)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 28**](#_Toc122062442)

[**ПРИЛОЖЕНИЕ А 29**](#_Toc122062443)

[**ПРИЛОЖЕНИЕ Б 60**](#_Toc122062444)

[**ПРИЛОЖЕНИЕ В 61**](#_Toc122062445)

[**ПРИЛОЖЕНИЕ Г 62**](#_Toc122062446)

# ВВЕДЕНИЕ

Адресная книга или имя и адресная книга (NAB )[1] - это книга или база данных , используемые для хранения записей, называемых контактов . Каждая контактная запись обычно состоит из нескольких стандартных полей (например: имя, фамилия, название компании, адрес , телефон номер, e -почта адрес, номер факса , номер мобильного телефона ). Большинство таких систем хранят детали в алфавитном порядке имен людей, хотя в бумажных -содержащих адресных книгах записи могут легко выйти из строя по мере того, как владелец вставляет данные о других людях или по мере того, как люди перемещаются. Во многих адресных книгах используются небольшие папки кольцевых папок , которые позволяют добавлять, удалять и перемещать страницы, чтобы освободить место.

Адресные книги также могут отображаться как программное обеспечение , разработанное для этой цели, например, «Адресная книга» , включенное в Apple Inc. Mac OS X . Простые адресные книги были включены в программное обеспечение электронной почты на протяжении многих лет, хотя в 1990-х годах и позже появились более продвинутые версии; а также в мобильных телефонах .

A менеджер личной информации (PIM) объединяет адресную книгу, календарь , список задач и иногда другие функции.

Записи можно импортировать и экспортировать из программного обеспечения, чтобы передавать их между программами или компьютерами. Общие форматы файлов для этих операций:

* JSON
* XML
* с разделителями табуляции (\* .tab, \* .txt)
* LDIF (\* .ldif, \* .ldi)
* через запятую (\*. csv)
* vCard (\* .vcf)

Отдельные записи часто передаются как vCard (\* .vcf), что примерно сопоставимо с физическим бизнесом. карты . А некоторые программные приложения, такие как Lotus Notes и Open Contacts, могут обрабатывать файл vCard, содержащий несколько записей vCard.

Интернет-адресная книга обычно позволяет пользователям создавать свои собственные веб-страницы (или страницы профиля), которые затем индексируются поисковыми системами, такими как Google и Yahoo. Это, в свою очередь, позволяет другим людям находить пользователей с помощью поиска по их имени, а затем связываться с ними через их веб-страницу, содержащую их личную информацию. Возможность найти людей, зарегистрированных в онлайн-адресных книгах, с помощью поисковой системы обычно зависит от распространенности имени и количества результатов по имени. Как правило, пользователи таких систем могут синхронизировать свои контактные данные с другими известными им пользователями, чтобы обеспечить актуальность их контактной информации.

У многих людей есть много разных адресных книг: их учетные записи электронной почты , их мобильный телефон и "списки друзей" на их социальные сети . Сетевая адресная книга позволяет им организовывать и управлять своими адресными книгами через единый интерфейс и делиться своими контактами в разных адресных книгах и социальных сетях.

На основании вышеизложенного определена тема курсового проектирования как «Адресная книга с поддержкой фотографий» с последующей разработкой программного средства «Contacts».

# 1. ОБЗОР ЛИТЕРАТУРЫ

В этом разделе речь пойдет о различных методах сохранения данных.

## 1.1 JSON

JSON— текстовый формат обмена данными, основанный на JavaScript. Как и многие другие текстовые форматы, JSON легко читается людьми. Формат JSON был разработан Дугласом Крокфордом[2].

Несмотря на происхождение от JavaScript, формат считается независимым от языка и может использоваться практически с любым языком программирования. Для многих языков существует готовый код для создания и обработки данных в формате JSON.

JSON-текст представляет собой (в закодированном виде) одну из двух структур:

* Набор пар ключ: значение. В различных языках это реализовано как запись, структура, словарь, хеш-таблица, список с ключом или ассоциативный массив. Ключом может быть только строка (регистрозависимость не регулируется стандартом, это остаётся на усмотрение программного обеспечения. Как правило, регистр учитывается программами — имена с буквами в разных регистрах считаются разными, например[5]), значением — любая форма. Повторяющиеся имена ключей допустимы, но не рекомендуются стандартом; обработка таких ситуаций происходит на усмотрение программного обеспечения, возможные варианты — учитывать только первый такой ключ, учитывать только последний такой ключ, генерировать ошибку.
* Упорядоченный набор значений. Во многих языках это реализовано как массив, вектор, список или последовательность.

Структуры данных, используемые JSON, поддерживаются любым современным языком программирования, что и позволяет применять JSON для обмена данными между различными языками программирования и программными системами.

В качестве значений в JSON могут быть использованы:

* запись — это неупорядоченное множество пар ключ:значение, заключённое в фигурные скобки «{ }». Ключ описывается строкой, между ним и значением стоит символ «:». Пары ключ-значение отделяются друг от друга запятыми.
* массив (одномерный) — это упорядоченное множество значений. Массив заключается в квадратные скобки «[ ]». Значения разделяются запятыми. Массив может быть пустым, то есть не содержать ни одного значения. Значения в пределах одного массива могут иметь разный тип.
* число (целое или вещественное).
* литералы true (логическое значение «истина»), false (логическое значение «ложь») и null.
* строка — это упорядоченное множество из нуля или более символов юникода, заключённое в двойные кавычки. Символы могут быть указаны с использованием escape-последовательностей, начинающихся с обратной косой черты «\» (поддерживаются варианты \", \\, \/, \t, \n, \r, \f и \b), или записаны шестнадцатеричным кодом в кодировке Unicode в виде \uFFFF.

Следующий пример показывает JSON-представление данных об объекте, описывающем человека. В данных присутствуют строковые поля имени и фамилии, информация об адресе и массив, содержащий список телефонов. Как видно из примера, значение может представлять собой вложенную структуру.

{

**"firstName"**: "Иван",

**"lastName"**: "Иванов",

**"address"**: {

**"streetAddress"**: "Московское ш., 101, кв.101",

**"city"**: "Ленинград",

**"postalCode"**: 101101

},

**"phoneNumbers"**: [

"812 123-1234",

"916 123-4567"

]

}

В качестве значений в JSON могут быть использованы как числа, так и строки. Поэтому запись "postalCode": "101101" содержит строку, а "postalCode": 101101 — уже числовое значение. Из-за слабой типизации в JavaScript и PHP строка может быть приведена к числу и не влиять на логику программы. Тем не менее, рекомендуется аккуратно обращаться с типом значения, так как JSON служит для межсистемного обмена.

## 1.2 XML

XML — «расширяемый язык разметки»[3]. Спецификация XML описывает XML-документы и частично описывает поведение XML-процессоров (программ, читающих XML-документы и обеспечивающих доступ к их содержимому). XML разрабатывался как язык с простым формальным синтаксисом, удобный для создания и обработки документов как программами, так и человеком, с акцентом на использование в Интернете. Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка.

Расширение XML — это конкретная грамматика, созданная на базе XML и представленная словарём тегов и их атрибутов, а также набором правил, определяющих, какие атрибуты и элементы могут входить в состав других элементов. Сочетание простого формального синтаксиса, удобства для человека, расширяемости, а также базирование на кодировках Юникод для представления содержания документов привело к широкому использованию как, собственно, XML, так и множества производных специализированных языков на базе XML в самых разнообразных программных средствах.

C физической точки зрения документ состоит из сущностей (англ. entities), из которых каждая может ссылаться на другую сущность. Единственный корневой элемент — документная сущность. Содержание сущностей — символы.

С логической точки зрения документ состоит из комментариев (англ. comments), объявлений (англ. declarations), элементов (англ. elements), ссылок на сущности (англ. character references) и инструкций обработки (англ. processing instructions). Всё это в документе структуризуется разметкой (англ. markup).

Сущность — мельчайшая часть в документе. Все сущности что-нибудь содержат, и у всех них есть имя (существуют исключения, напр. документная сущность). Проще говоря, термин «сущность» описывает «сущую вещь», «что-то».

Документ состоит из сущностей, содержание которых — символы. Все символы разделены на два типа: символы данных (англ. character data) и символы разметки. К разметке относятся:

* теги (англ. tags) <- обозначают границы элементов
* объявления и инструкции обработки, включая их атрибуты (англ. attributes)
* ссылки на сущности
* комментарии
* а также последовательности символов, обрамляющие секции «CDATA»

Часть документа, не принадлежащая разметке, составляет символьные данные документа.

На языке XML структура из прошлого раздела выглядела бы примерно так:

**<person>**

**<firstName>**Иван**</firstName>**

**<lastName>**Иванов**</lastName>**

**<address>**

**<streetAddress>**Московское ш., 101, кв.101**</streetAddress>**

**<city>**Ленинград**</city>**

**<postalCode>**101101**</postalCode>**

**</address>**

**<phoneNumbers>**

**<phoneNumber>**812 123-1234**</phoneNumber>**

**<phoneNumber>**916 123-4567**</phoneNumber>**

**</phoneNumbers>**

**</person>**

или так:

**<person** firstName="Иван" lastName="Иванов"**>**

**<address** streetAddress="Московское ш., 101, кв.101" city="Ленинград" postalCode="101101" **/>**

**<phoneNumbers>**

**<phoneNumber>**812 123-1234**</phoneNumber>**

**<phoneNumber>**916 123-4567**</phoneNumber>**

**</phoneNumbers>**

**</person>**

Для выполнения данной задачи был выбран формат файла JSON.

Основная причина выбора данного формата файла является то, что он удобен как в чтении и создании файлов человеком, так и работой с этим форматом в современных языках программирования.

В данной курсовой работе нам нужен формат файла, который :

* занимает меньше места
* удобен в изменении, создании и работе с файлами на языке программирования с++ на фреймворке Qt
* может хранить числа и строки
* удобочетаем для человека

Так как мы будем хранить информацию о контактах человека, а их может быть большое количество( примерно от 130 до 200),то один из ключевых факторов в выборе формата файла для сохранения в него данных это компактность(файл занимает меньше места чем аналоги).

Второй ключевой фактор – это сложность работы с этим форматом на языке c++ с использованием фреймворка Qt.

В фреймворке Qt есть инструменты как для работы с форматом JSON, так и с XML. Они представляют собой различные классы для чтения, записи, изменения, просмотра и парсинга JSON и XML файлов. В данном фреймворке работа с JSON и XML форматами одинаково проста с точки зрения разработки адресной книги[4].

Ключевое различие между JSON и XML заключается в том, что JSON имеет меньший размер файла по сравнению с XML[5]. JSON обращается к данным через объекты JSON, тогда как XML требует анализа данных. JSON легко читается, поскольку имеет более организованную структуру кода. С другой стороны, XML трудно интерпретировать из-за его сложной структуры.

Так же формат JSON намного легче читается для человека, чем данные в виде XML формата, что видно на примере одной и той же структуры на JSON

{

**"firstName"**: "Иван",

**"lastName"**: "Иванов",

**"address"**: {

**"streetAddress"**: "Московское ш., 101, кв.101",

**"city"**: "Ленинград",

**"postalCode"**: 101101

},

**"phoneNumbers"**: [

"812 123-1234",

"916 123-4567"

]

}

И в виде XML :

**<person>**

**<firstName>**Иван**</firstName>**

**<lastName>**Иванов**</lastName>**

**<address>**

**<streetAddress>**Московское ш., 101, кв.101**</streetAddress>**

**<city>**Ленинград**</city>**

**<postalCode>**101101**</postalCode>**

**</address>**

**<phoneNumbers>**

**<phoneNumber>**812 123-1234**</phoneNumber>**

**<phoneNumber>**916 123-4567**</phoneNumber>**

**</phoneNumbers>**

**</person>**

Подводя итог, JSON был выбран по причине хорошей читаемости для человека(что может помочь при проблемах с повреждениями файла и др.) а также занимает меньше места по сравнению с XML.

# 2.СТРУКТУРНОЕ ПРОЕКТРИРОВАНИЕ

Программный код был разбит на модули для более удобного использования и изменения функционала программы. Для подробного описания классов смотрите UML-диаграмму в Приложении В.

## 2.1 main.cpp

В этом файле написана функция main, которая запускает приложение с помощью класса QAllpication. После этого создается объект класса MainWindow, который отвечает за интерфейс приложения, и вызывается его метод show(), который отображает интерфейс приложения.

## 2.2 mainwindow.h

Данный файл отвечает за интерфейс приложения. Создает верхнее меню для работы с файлами и для работы с контактами. Класс содержит поля типа AddressWidget \*, который отображает таблицы и ее вкладки, и QAction \*, нужный для корректной работы меню.

## 2.3 mainwindow.cpp

Файл с реализацией методов класса MainWindow, а точнее конструктора, createMenus(), openFile(), saveFile() и updateActions(const QItemSelection &selection), которые отвечают за создание меню и вызов дальнейших методов, вызов диалога выбора файла для чтения или записи и обновления статуса функций поиска и удаления контактов в зависимости от того, пустой ли список контактов или нет.

## 2.4 addresswidget.h

Файл описывающий класс AddressWidget, отвечающий за взаимодействие между интерфейсом и данными , основную логику программы и отображение таблиц на экране. Содержит методы для чтения и записи файлов, добавления или изменения, поиска и удаления данных в таблице.

## 2.5 addresswidget.cpp

Файл содержит реализации методов класса AddressWidget. Методы добавления в поиска контактов вызывают соответствующие диалоговые окна и с помощью методов класса TableModel соответствующе манипулируют данными списка контактов. При отсутствии контактов создает специальную вкладку класса NewAddressTab, мотивирующую пользователя добавить первый контакт. Конструктор дает пользователю возможность закрывать вкладки с таблицами, за исключением вкладки со всеми контактами.

## 2.6 tablemodel.h

Данный файл содержит структуру Contact, содержащую данную о контакте и методы сравнения двух контактов и записи контакта в объект JSON.

Также описывает класс TableModel, наследующийся от класса QAbstractTableModel. Класс содержит контейнер List<Contact>, содержащий данные всех контактов, и методы манипуляции и просмотра данных.

## 2.7 tablemodel.cpp

Содержит реализацию методов класса TableModel. Методы класса реализуют функции добавления, изменения, удаления и просмотра данных контейнера contacts.

## 2.8 exception.h

Описывает класс Exception и 4 дочерних класса : ArgumentException, MemoryException, IndexException, EmptyListException. У каждого класса есть два приватных поля errorCode и message, хранящих код ошибки и сообщение, которое описывает ошибку. Дочерние классы отличаются от родительского только кодом ошибки :

* C0000 – Exception
* C0001 – ArgumentException
* C0002 – MemoryException
* C0003 – IndexException
* C0004 – EmptyListException

## 2.9 list.h

Файл содержащий описание структуры Element, контейнера List и итератора к нему, также содержит реализацию методов итератора.

Структура представляет собой элемент контейнера, содержащий данные и указатели на предыдущий и следующий элементы.

Контейнер содержит 3 приватных поля : начало списка, конец списка и количество его элементов.

## 2.10 list.cpp

Файл содержит реализацию методов контейнера List. При ошибках выделения памяти, индекса, пустого контейнера кидает соответствующие исключения.

## 2.11 adddialog.h

Описывает класс AddDialog, который наследуется от класса QDialog. Класс содержит приватные поля nameText, addressText, emailText, picturePath, pictureButton. Методы класса возвращают данные которые пользователь ввел. Класс реализует диалог для добавления и изменения контакта.

## 2.12 adddialog.cpp

Содержит реализацию методов класса AddDialog. Конструктор создает разметку для полей и надписей в диалоге. Метод editAddress вызывается при изменении контакта и прописывает данные контакта до их изменения, также не позволяет изменять имя контакта, так как при изменении этого поля контакт по сути будет уже другим контактом.

## 2.13 errordialog.h

Описывает класс ErrorDialog, который наследуется от класса QDialog. Класс не содержит приватные поля . Метод у класса только один – конструктор . Класс реализует диалог который выводит код и сообщение ошибки, если такое повляется.

## 2.14 errordialog.cpp

Файл содержит реализацию конструктора класса ErrorDialog. Конструктор принимает ссылку на родительски й виджет и исключение, которое должен вывести.

## 2.15 finddialog.h

Описывает класс FindDialog, который наследуется от класса QDialog. Класс содержит приватные поля nameText, addressText, emailText. Методы класса возвращают данные которые пользователь ввел. Класс реализует диалог для поиска контакта по имени, адресу и(или) электронной почте.

## 2.16 finddialog.cpp

Содержит реализацию методов класса FindDialog. Конструктор создает разметку для полей и надписей в диалоге.

# 3. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Для подробного рассмотрения кода и понимания программы ниже представлены алгоритмы по шагам основных функций и диаграмма последовательности. Также в приложении Г представлены блок-схемы функций.

## 3.1 Диаграмма последовательности

Диаграмма последовательности — UML-диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл объекта (создание-деятельность-уничтожение некой сущности) и взаимодействие актеров (действующих лиц) информационной системы в рамках прецедента. Диаграмма последовательности разработанного приложения представлена на рисунке 3.1.

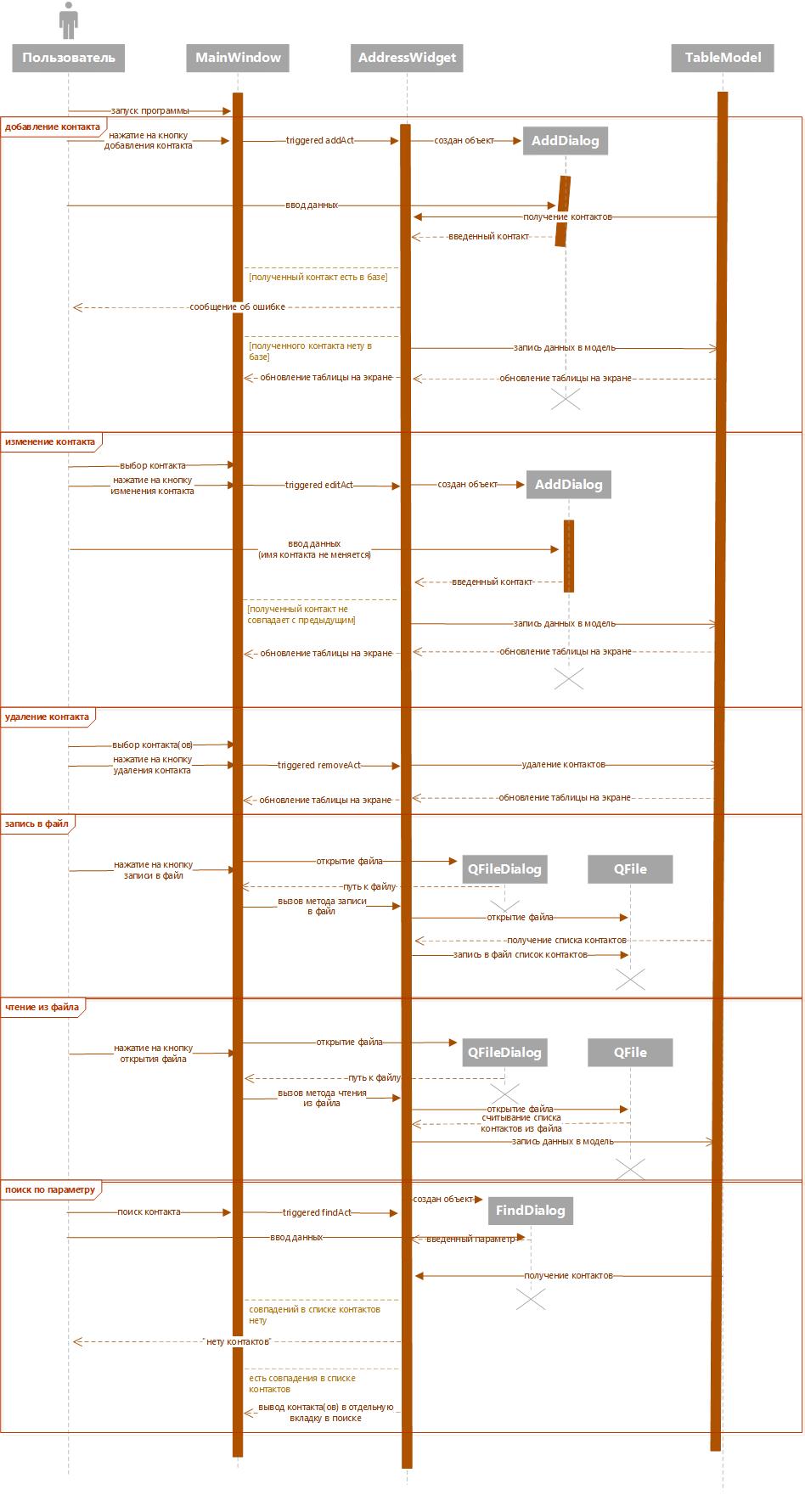


Рисунок 3 1

## 3.2 Метод вставки в контейнер элемента void List<T>::Insert(T \_data, int index)

Шаг 1. Проверка, корректна ли позиция;

Шаг 2. Проверка, вставка ли в конец списка (по указателю end)

Шаг 3. Проверка, вставка ли в начало списка (перед begin)

Шаг 4. Получить элемент перед позицией вставки

Шаг 5. Получить элемент в позиции вставки

Шаг 6. Создать новый элемент и вставить его в список

Шаг 7. Увеличить количество элементов

## 3.3 Метод удаления контакта void AddressWidget::removeEntry()

Шаг 1. Инициализация QTableView \*temp;

Шаг 2. Инициализация QSortFilterProxyModel \*proxy = temp->model();

Шаг 3. QItemSelectionModel \*selectionModel = temp->selectionModel();

Шаг 4. Инициализация QModelIndexList indexes = selectionModel->selectedRows();

Шаг 5. Начать цикл QModelIndex index : indexes

Шаг 6. Инициализация int row = proxy->mapToSource(index).row();

Шаг 7. Вызов метода удаления строки table->removeRows(row, 1, QModelIndex());

Шаг 8. Конец цикла;

Шаг 9. Если количество строк в таблице равно 0, то вставка вкладки newAddressTab;

## 3.4 Метод вставки строк в таблицу bool TableModel:: insertRows(int position, int rows, const QModelIndex &index)

Шаг 1. Вызов метода beginInsertRows(QModelIndex(), position, position + rows - 1);

Шаг 2. Начать цикл по переменной int row = 0; row < rows; ++row;

Шаг 3. Вызов метода вставки строки в лист;

Шаг 4. Конец цикла;

Шаг 5. Вызов метода endInsertRows();

Шаг 6. Возвращаем значение true;

## 3.5 Метод удаления строки из таблицы bool TableModel::removeRows(int position, int rows, const QModelIndex &index)

Шаг 1. Вызов метода beginRemoveRows(QModelIndex(), position, position + rows - 1);

Шаг 2. Начать цикл по переменной int row = 0; row < rows; ++row;

Шаг 3. Вызов метода удаления строки из листа контактов по позиции;

Шаг 4. Конец цикла;

Шаг 5. Вызов метода endRemoveRows();

Шаг 6. Возвращаем значение true;

## 3.6 Метод записи контакта в объект JSON void Contact::toJson(QJsonObject &json) const

Шаг 1. Создаем поле «name» и присваиваем ему значение name;

Шаг 2. Создаем поле «address» и присваиваем ему значение address;

Шаг 3. Создаем поле «email» и присваиваем ему значение email;

Шаг 4. Создаем поле «picture path» и присваиваем ему значение picturePath;

# *4. РУКОВОДСТОВО ПОЛЬЗОВАТЕЛЯ*

Программа предназначена для ведения адресной книги с поддержкой фотографий. Реализованы следующие функции : добавление, редактирование, поиск по имени/адресу/электронной почте и удаление контаков, чтение и запись в файл списка контактов.

Контакт содержит в себе 4 поля : имя, адрес, электронную почту и путь к его фотографии.

При запуске программы появляется окно программы с двумя верхними меню : File, Tools и две вкладки : Address Book и ALL. Окно программы при первом запуске показано на рисунке 4.1

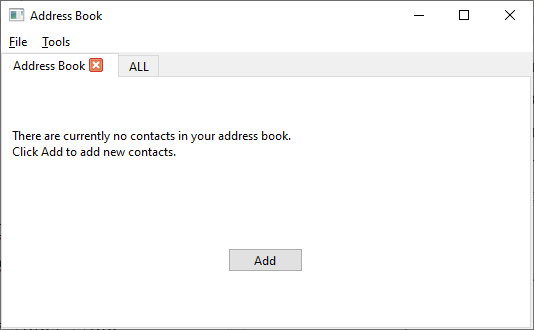


Рисунок 4. 1 Первый запуск программы

## 4.1 Добавление контакта

Добавить новые контакты можно следующими способами :

* Через пункт “New Entry…” в меню Tools
* Нажать комбинацию клавиш Ctrl + N, которая вызывает пункт меню “New Entry…”
* Нажать на кнопку Add в середине вкладки Address Book, которая появляется если список контактов пуст.
* Через пункт “Open…” меню File прочитать файл формата JSON
* Нажать компбинацию клавиш Ctrl + O, которая вызывает пункт меню “Open…”

В каждом из случаев появляется диалоговое окно для добавления нового контакта. Окно состоит из трех полей ввода с названиями, кнопки добавления картинки и кнопок “OK” и “Cancel”. При нажатии на кнопку добавления картинки появляется стандартный проводник Windows для выбора файла формата .png, .xmp или .jpg . При попытке добавить контакт с одним или более пустых полей, показывается диалоговое окно ошибки. Диалоговое окно добавления контакта показано на рисунке 4.2 . Диалоговое окно ошибки показано на рисунке 4.3 .

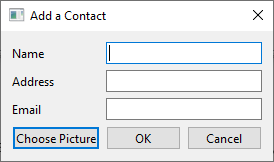


Рисунок 4. 2 Добавление контакта

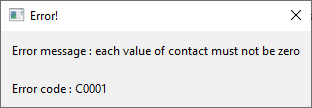


Рисунок 4. 3 Ошибка

После добавления первого контакта вкладка Address Book автоматически закрывается и остается только вкладка ALL со всеми контактами. Она представляет собой таблицу с 4 столбцами : Name, Address, Email, Photo. Каждая строка представляет собой отдельно взятого контакта . Таблица контактов показана на рисунке 4.4 .

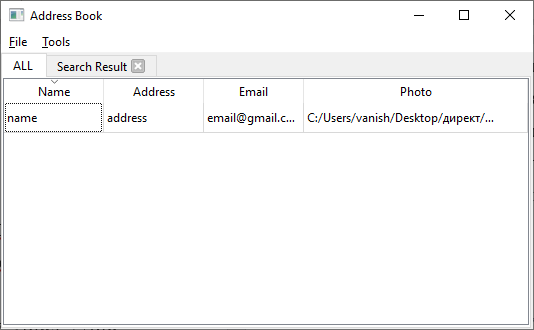


Рисунок 4. 4 таблица контактов

Чтобы просмотреть фото контакта, требуется навести курсор мыши на поле photo конкретного контакта, после чего появится фото. Пример показан на рисунке 4.5

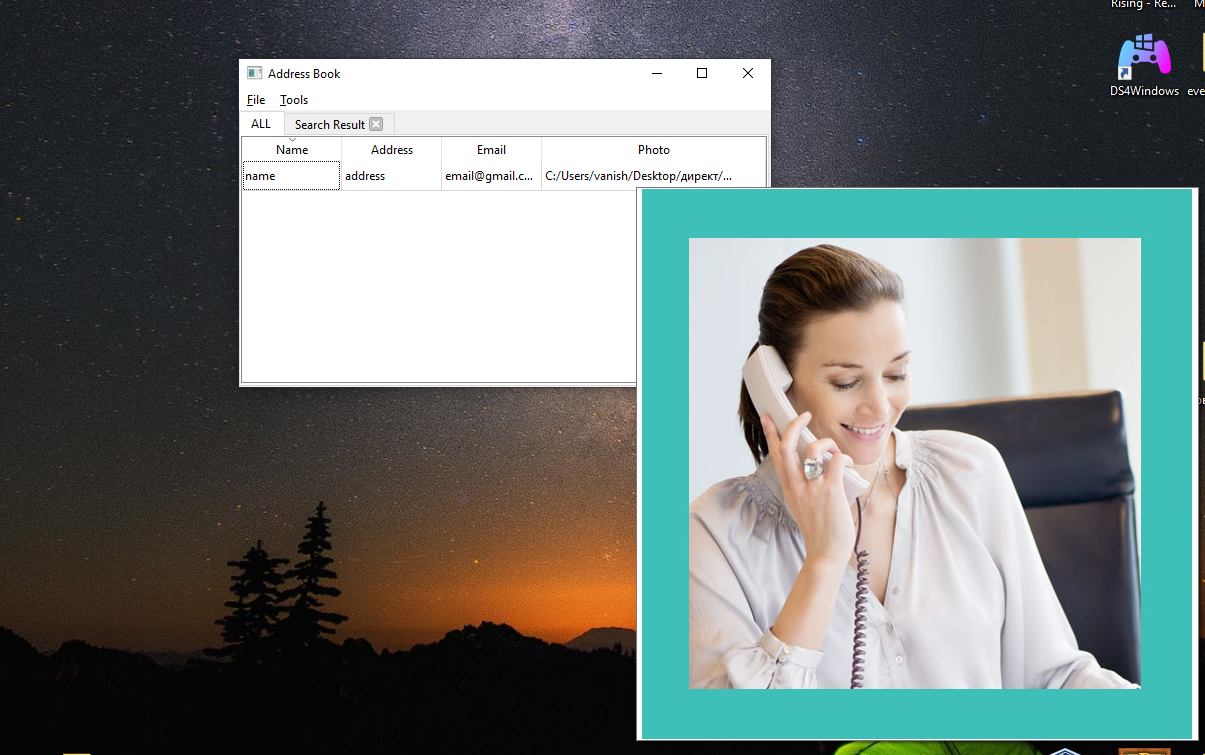


Рисунок 4. 5

## 4.2 Редактирование конаткта

Чтобы изменить контакт, надо сначала нажать на него, потом в меню Tools нажать на “Edit Entry…” либо нажать комбинацию клавиш Ctrl + E. При нажатии появляется тоже диалоговое окно, что и при добавлении, только данные изменяемого контакта уже вписаны в поля, также поле name становится некликабельным, чтобы пользователь не смог его изменить. Окно редактирования контакта показано на рисунке 4.6 .

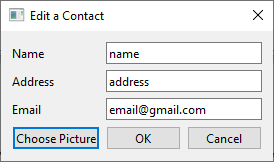


Рисунок 4. 6 окно редактирования контакта

## 4.3 Фильтр контактов

Чтобы отфильтровать контакты по каким-либо полям и вывести их в отдельную вкладку, надо в меню Tools нажать на “Find Entry…” либо нажать комбинацию клавиш Ctrl + F. После этого появляется диалоговое окно поиска с 3 полями : name, address и email. Окно поиска показано на рисунке 4.7 .

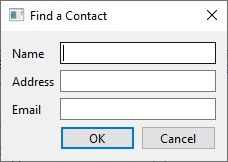


Рисунок 4. 7 диалоговое окно поиска

Чтобы отфильтровать по какому-либо их этих параметров впишите в соответствующее поле это значение, после чего нажмите OK. После этого программа выведет в отдельную вкладку Search Result отфильтрованный список контактов. Вкладка Search Result показана на рисунке 4.8 .

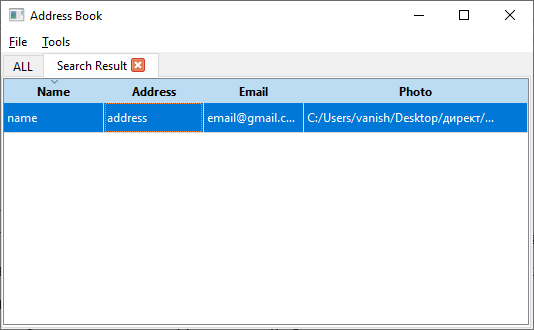


Рисунок 4. 8 вкладка с результатами поиска

Выведенные контакты фильтруются по всем полям. Если одно или несколько полей фильтра оставить пустым, то выведутся все контакты, подходящие по остальным критериям. Если все поля оставить пустыми, то выведутся все контакты.

## 4.4 Удаление контакта

Чтобы удалить контакт, надо выбрать контакт, который вы хотите удалить, и в меню Tools нажать “Remove Entry…” или нажать клавишу Del.

## 4.5 Запись в файл

Чтобы сохранить список контактов, с которым вы работаете в данный момент, в файл, надо открыть меню File и нажать “Save as…” либо нажать комбинацию клавиш Ctrl + S . После этого откроется проводник windows в котором можно либо выбрать файл с расшрением .json, либо создать его. Выбор файла показан на рисунке 4.9 .

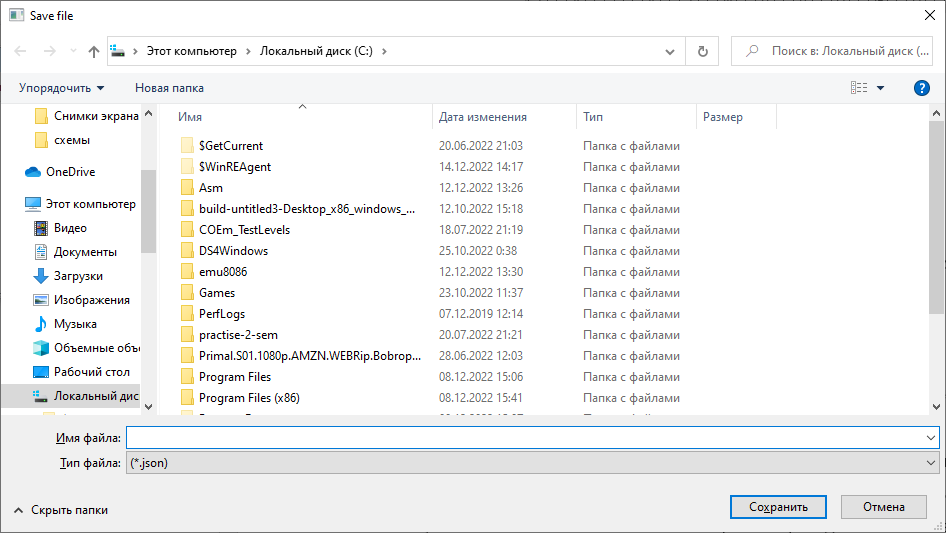


Рисунок 4. 9 выбор файла для записи

Если вы выбрали файл, то программа запишет в него весь список контактов в формате JSON.

## 4.6 Чтение из файла

Если у вас уже есть подходящий JSON файл с списком контактов, то вы можете его прочитать и занести контакты в программу. Для этого вам надо в меню File выбрать “Open…” либо нажать комбинацию клавиш Ctrl + O. После этого откроется проводник windows в котором нужно выбрать файл с расшрением .json.

Если файл корректен, то данные из него внесутся в программу.

Файл должен быть с расширением .json чтобы его прочитать. Также он должен иметь определенную структуру. Файл должен содержать объект “contacts” содержащий массив объектов, у каждого из которых есть только 4 поля : “name”, “address”, “email”, “picture path”. Каждое из этих полей должно быть типа String. Пример корректного файла :

"contacts": [

{

"address": "",

"email": "rqew",

"name": "fsdsfd",

"picture path": ""

},

{

"address": "rewewr",

"email": "",

"name": "dsffd",

"picture path": "C:/Users/vanish/Pictures/Screenshots/Снимок экрана (1).png"

},

{

"address": "address",

"email": "email",

"name": "name",

"picture path": "C:/Users/vanish/Pictures/Screenshots/Снимок экрана (1).png"

}

]

}

# ЗАКЛЮЧЕНИЕ

В данном курсовом проекте была реализована программа Contacts на основе выученного материала и знаний по дисциплине «Программирование на языках высокого уровня». В данной курсовой затрагивались такие темы из дисциплины «Программирование на языках высокого уровня», как : классы, работа с файлами, шаблоны и классы контейнеры, итераторы, обработка исключений.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Адресная книга – Address Book [Электронный ресурс]. – Режим доступа : <https://wiki5.ru/wiki/Address_book> .

[2] JSON [Электронный ресурс]. – Режим доступа : <https://ru.wikipedia.org/wiki/JSON> .

[3] XML [Электронный ресурс]. – Режим доступа : <https://ru.wikipedia.org/wiki/XML> .

[4] JSON Support in Qt [Электронный ресурс] . – Режим доступа : <https://doc.qt.io/qt-5/json.html> .

[5] JSON или XML — в чем разница? [Электронный ресурс] . – Режим доступа : <https://clck.ru/333fVn> .

# ПРИЛОЖЕНИЕ А

*(обязательное)*

Листинг кода

\\ файл “main.cpp”

#include "mainwindow.h"

#include <QApplication>

int **main**(int **argc**, char \***argv**[])

{

QApplication **a**(argc, argv);

MainWindow **w**;

w.show();

return a.exec();

}

\\ файл “"mainwindow.h”

#ifndef MAINWINDOW\_H

#define **MAINWINDOW\_H**

#include "addresswidget.h"

#include <QMainWindow>

//! [0]

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

**public**:

MainWindow();

private slots:

void updateActions(const QItemSelection &selection);

void **openFile**();

void **saveFile**();

private:

void **createMenus**();

AddressWidget \***addressWidget**;

QAction \***editAct**;

QAction \***removeAct**;

};

//! [0]

#endif // MAINWINDOW\_H

\\ файл “"mainwindow.сpp”

#include "mainwindow.h"

#include <QAction>

#include <QFileDialog>

#include <QMenuBar>

MainWindow::**MainWindow**()

: QMainWindow(),

addressWidget(new AddressWidget)

{

setCentralWidget(addressWidget);

createMenus();

setWindowTitle(tr("Address Book"));

}

void MainWindow::**createMenus**()

{

QMenu \***fileMenu** = menuBar()->addMenu(tr("&File"));

QAction \***openAct** = new QAction(tr("&Open..."), this);

openAct->setShortcut(QKeySequence::Open);

fileMenu->addAction(openAct);

connect(openAct, &QAction::triggered, this, &MainWindow::openFile);

//! [1a]

QAction \***saveAct** = new QAction(tr("&Save As..."), this);

saveAct->setShortcut(QKeySequence::Save);

fileMenu->addAction(saveAct);

connect(saveAct, &QAction::triggered, this, &MainWindow::saveFile);

fileMenu->addSeparator();

QAction \***exitAct** = new QAction(tr("E&xit"), this);

exitAct->setShortcut(QKeySequence::Close);

fileMenu->addAction(exitAct);

connect(exitAct, &QAction::triggered, this, &QWidget::close);

QMenu \***toolMenu** = menuBar()->addMenu(tr("&Tools"));

QAction \***addAct** = new QAction(tr("&New Entry..."), this);

addAct->setShortcut(QKeySequence::New);

toolMenu->addAction(addAct);

connect(addAct, &QAction::triggered,

addressWidget, &AddressWidget::showAddEntryDialog);

QAction \***findAct** = new QAction(tr("&Find Entry..."), this);

findAct->setShortcut(QKeySequence::Find);

toolMenu->addAction(findAct);

connect(findAct, &QAction::triggered, addressWidget, &AddressWidget::findEntry);

editAct = new QAction(tr("&Edit Entry..."), this);

editAct->setShortcut(QKeySequence(tr("Ctrl+E")));

editAct->setEnabled(false);

toolMenu->addAction(editAct);

connect(editAct, &QAction::triggered, addressWidget, &AddressWidget::editEntry);

toolMenu->addSeparator();

removeAct = new QAction(tr("&Remove Entry"), this);

removeAct->setShortcut(QKeySequence::Delete);

removeAct->setEnabled(false);

toolMenu->addAction(removeAct);

connect(removeAct, &QAction::triggered, addressWidget, &AddressWidget::removeEntry);

connect(addressWidget, &AddressWidget::selectionChanged,

this, &MainWindow::updateActions);

}

void MainWindow::**openFile**()

{

QString **fileName** = QFileDialog::getOpenFileName(this, tr("Open file"), tr(""), tr("(\*.json)"));

if (!fileName.isEmpty())

addressWidget->readFromFile(fileName);

}

void MainWindow::**saveFile**()

{

QString **fileName** = QFileDialog::getSaveFileName(this, tr("Save file"), tr(""), tr("(\*.json)"));

if (!fileName.isEmpty())

addressWidget->writeToFile(fileName);

}

void MainWindow::**updateActions**(const QItemSelection &**selection**)

{

QModelIndexList **indexes** = selection.indexes();

if (!indexes.isEmpty()) {

removeAct->setEnabled(true);

editAct->setEnabled(true);

} else {

removeAct->setEnabled(false);

editAct->setEnabled(false);

}

}

\\ файл “"addresswidget.h”

#ifndef ADDRESSWIDGET\_H

#define **ADDRESSWIDGET\_H**

#include "newaddresstab.h"

#include "tablemodel.h"

#include <QItemSelection>

#include <QTabWidget>

QT\_BEGIN\_NAMESPACE

class QSortFilterProxyModel;

class **QItemSelectionModel**;

QT\_END\_NAMESPACE

//! [0]

class AddressWidget : public QTabWidget

{

Q\_OBJECT

public:

AddressWidget(QWidget \*parent = nullptr);

void readFromFile(const QString &fileName);

void writeToFile(const QString &fileName);

public slots:

void showAddEntryDialog();

void addEntry(const QString &name, const QString &address, const QString &email, const QString &picturePath);

void editEntry();

void findEntry();

void removeEntry();

signals:

void selectionChanged (const QItemSelection &selected);

private:

void setupTabs();

TableModel \*table;

NewAddressTab \*newAddressTab;

};

//! [0]

#endif // ADDRESSWIDGET\_H

\\ файл “"addresswidget.cpp”

#include"addresswidget.h"

#include "finddialog.h"

#include "adddialog.h"

#include "errordialog.h"

#include <QtWidgets>

AddressWidget::**AddressWidget**(QWidget \***parent**)

: QTabWidget(parent),

table(new TableModel(this)),

newAddressTab(new NewAddressTab(this))

{

connect(newAddressTab, &NewAddressTab::sendDetails,

this, &AddressWidget::addEntry);

connect(this->tabBar(), &QTabBar::tabCloseRequested, this->tabBar(), &QTabBar::removeTab);

addTab(newAddressTab, tr("Address Book"));

AddressWidget::setTabsClosable(true);

setupTabs();

}

void AddressWidget::**showAddEntryDialog**()

{

AddDialog **aDialog**;

if (aDialog.exec())

addEntry(aDialog.name(), aDialog.address(), aDialog.email(), aDialog.getPicturePath());

}

void AddressWidget::**addEntry**(const QString &**name**, const QString &**address**, const QString &**email**, const QString &**picturePath**)

{

try {

if(name == QString() || address == QString() || email == QString() || picturePath == QString())

throw ArgumentException("each value of contact must not be zero\n");

if (!table->getContacts().Contains({name, address, email, picturePath})) {

table->insertRows(0, 1, QModelIndex());

QModelIndex **index** = table->index(0, 0, QModelIndex());

table->setData(index, name, Qt::EditRole);

index = table->index(0, 1, QModelIndex());

table->setData(index, address, Qt::EditRole);

index = table->index(0, 2, QModelIndex());

table->setData(index, email, Qt::EditRole);

index = table->index(0, 3, QModelIndex());

table->setData(index, picturePath, Qt::EditRole);

if(indexOf(newAddressTab) != -1)

{

removeTab(indexOf(newAddressTab));

}

} else {

QMessageBox::information(this, tr("Duplicate Name"),

tr("The name \"%1\" already exists.").arg(name));

}

} catch (ArgumentException **ex**) {

qDebug() << "Error message: " << ex.what();

qDebug() << "Error code: " << ex.GetErrorCode();

ErrorDialog **errorDialog**(this, ex);

errorDialog.exec();

}

}

void AddressWidget::**findEntry**()

{

FindDialog **fDialog**;

if(fDialog.exec())

{

auto **nameProxyModel** = new QSortFilterProxyModel(this);

nameProxyModel->setSourceModel(table);

auto **addressProxyModel** = new QSortFilterProxyModel(this);

addressProxyModel->setSourceModel(table);

auto **emailProxyModel** = new QSortFilterProxyModel(this);

emailProxyModel->setSourceModel(table);

if(fDialog.name() != "" && fDialog.name() != nullptr)

{

const auto **regExp** = QRegularExpression(QLatin1StringView("[%1]").arg(fDialog.name()),

QRegularExpression::CaseInsensitiveOption);

nameProxyModel->setFilterRegularExpression(regExp);

nameProxyModel->setFilterKeyColumn(0);

addressProxyModel->setSourceModel(nameProxyModel);

emailProxyModel->setSourceModel(nameProxyModel);

}

if(fDialog.address() != "" && fDialog.address() != nullptr)

{

const auto **regExp** = QRegularExpression(QLatin1StringView("[%1]").arg(fDialog.address()),

QRegularExpression::CaseInsensitiveOption);

addressProxyModel->setFilterRegularExpression(regExp);

addressProxyModel->setFilterKeyColumn(1);

emailProxyModel->setSourceModel(addressProxyModel);

}

if(fDialog.email() != "" && fDialog.email() != nullptr)

{

const auto **regExp** = QRegularExpression(QLatin1StringView("[%1]").arg(fDialog.email()),

QRegularExpression::CaseInsensitiveOption);

emailProxyModel->setFilterRegularExpression(regExp);

emailProxyModel->setFilterKeyColumn(2);

}

QTableView \***tableView** = new QTableView;

tableView->setModel(emailProxyModel);

tableView->setSelectionBehavior(QAbstractItemView::SelectRows);

tableView->horizontalHeader()->setStretchLastSection(true);

tableView->verticalHeader()->hide();

tableView->setEditTriggers(QAbstractItemView::NoEditTriggers);

tableView->setSelectionMode(QAbstractItemView::SingleSelection);

tableView->setSortingEnabled(true);

connect(tableView->selectionModel(), &QItemSelectionModel::selectionChanged,

this, &AddressWidget::selectionChanged);

connect(this, &QTabWidget::currentChanged, this, [this, tableView](int tabIndex) {

if (widget(tabIndex) == tableView)

emit selectionChanged(tableView->selectionModel()->selection());

});

addTab(tableView, tr("Search Result"));

setCurrentWidget(tableView);

}

}

void AddressWidget::**editEntry**()

{

QTableView \***temp** = static\_cast<QTableView\*>(currentWidget());

QSortFilterProxyModel \***proxy** = static\_cast<QSortFilterProxyModel\*>(temp->model());

QItemSelectionModel \***selectionModel** = temp->selectionModel();

const QModelIndexList **indexes** = selectionModel->selectedRows();

QString **name**;

QString **address**;

QString **email**;

QString **picturePath**;

int **row** = -1;

for (const QModelIndex &index : indexes) {

row = proxy->mapToSource(index).row();

QModelIndex nameIndex = table->index(row, 0, QModelIndex());

QVariant varName = table->data(nameIndex, Qt::DisplayRole);

name = varName.toString();

QModelIndex addressIndex = table->index(row, 1, QModelIndex());

QVariant varAddr = table->data(addressIndex, Qt::DisplayRole);

address = varAddr.toString();

QModelIndex emailIndex = table->index(row, 2, QModelIndex());

QVariant varEmail= table->data(emailIndex, Qt::DisplayRole);

email = varEmail.toString();

QModelIndex picturePathIndex = table->index(row, 3, QModelIndex());

QVariant varPicturePath= table->data(picturePathIndex, Qt::DisplayRole);

picturePath = varPicturePath.toString();

}

AddDialog **aDialog**;

aDialog.setWindowTitle(tr("Edit a Contact"));

aDialog.editAddress(name, address, email, picturePath);

if (aDialog.exec()) {

const QString **newAddress** = aDialog.address();

if (newAddress != address) {

const QModelIndex **index** = table->index(row, 1, QModelIndex());

table->setData(index, newAddress, Qt::EditRole);

}

const QString **newEmail**= aDialog.email();

if (newEmail != email) {

const QModelIndex **index** = table->index(row, 2, QModelIndex());

table->setData(index, newEmail, Qt::EditRole);

}

const QString **newPicturePath**= aDialog.getPicturePath();

if (newPicturePath!= picturePath) {

const QModelIndex **index** = table->index(row, 3, QModelIndex());

table->setData(index, newPicturePath, Qt::EditRole);

}

}

}

void AddressWidget::**removeEntry**()

{

QTableView \***temp** = static\_cast<QTableView\*>(currentWidget());

QSortFilterProxyModel \***proxy** = static\_cast<QSortFilterProxyModel\*>(temp->model());

QItemSelectionModel \***selectionModel** = temp->selectionModel();

const QModelIndexList **indexes** = selectionModel->selectedRows();

for (QModelIndex index : indexes) {

int row = proxy->mapToSource(index).row();

table->removeRows(row, 1, QModelIndex());

}

if (table->rowCount(QModelIndex()) == 0)

insertTab(0, newAddressTab, tr("Address Book"));

}

void AddressWidget::**setupTabs**()

{

using namespace Qt::StringLiterals;

auto **proxyModel** = new QSortFilterProxyModel(this);

proxyModel->setSourceModel(table);

proxyModel->setFilterKeyColumn(0);

QTableView \***tableView** = new QTableView;

tableView->setModel(proxyModel);

tableView->setSelectionBehavior(QAbstractItemView::SelectRows);

tableView->horizontalHeader()->setStretchLastSection(true);

tableView->verticalHeader()->hide();

tableView->setEditTriggers(QAbstractItemView::NoEditTriggers);

tableView->setSelectionMode(QAbstractItemView::SingleSelection);

tableView->setSortingEnabled(true);

connect(tableView->selectionModel(), &QItemSelectionModel::selectionChanged,

this, &AddressWidget::selectionChanged);

connect(this, &QTabWidget::currentChanged, this, [this, tableView](int tabIndex) {

if (widget(tabIndex) == tableView)

emit selectionChanged(tableView->selectionModel()->selection());

});

addTab(tableView, "ALL");

this->tabBar()->tabButton(1, QTabBar::RightSide)->resize(0,0);

}

void AddressWidget::**readFromFile**(const QString &**fileName**)

{

QFile **file**( fileName );

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

{

QByteArray **bytes** = file.readAll();

file.close();

QJsonParseError **jsonError**;

QJsonDocument **document** = QJsonDocument::fromJson( bytes, &jsonError );

if( jsonError.error != QJsonParseError::NoError )

{

return ;

}

if( document.isObject() )

{

QJsonArray **jsonContacts** = document.object().value("contacts").toArray();

for(int i = 0 ; i < jsonContacts.count() ; i++)

{

QJsonObject json = jsonContacts.at(i).toObject();

if(json.count() == 4 && json.contains("name") && json.contains("address") && json.contains("email") && json.contains("picture path"))

{

if(json.value("name").isString() && json.value("address").isString() && json.value("email").isString() && json.value("picture path").isString())

{

addEntry(json.value("name").toString(), json.value("address").toString(), json.value("email").toString(), json.value("picture path").toString());

}

}

}

}

}

}

void AddressWidget::**writeToFile**(const QString &**fileName**)

{

QJsonArray **contactsArray**;

for (int **i** = 0; i < table->getContacts().Count(); i++)

{

QJsonObject **jsonContact**;

table->getContacts().GetElement(i).toJson(jsonContact);

contactsArray.append(jsonContact);

}

QJsonObject **jsonContacts**;

jsonContacts.insert("contacts", contactsArray);

QJsonDocument **document**;

document.setObject( jsonContacts);

QByteArray **bytes** = document.toJson( QJsonDocument::Indented );

QFile **file**( fileName );

try {

if( !file.open( QIODevice::WriteOnly | QIODevice::Text | QIODevice::Truncate ) )

{

throw Exception("Failed to open file\n");

}

QTextStream **iStream**( &file );

iStream << bytes;

file.close();

} catch (Exception **ex**) {

qDebug() << "Error message: " << ex.what();

qDebug() << "Error code: " << ex.GetErrorCode();

ErrorDialog **errorDialog**(this, ex);

errorDialog.exec();

}

}

\\ файл “"tablemodel.h”

#ifndef TABLEMODEL\_H

#define **TABLEMODEL\_H**

#include "list.h"

#include "list.cpp"

#include <QAbstractTableModel>

#include <QPixMap>

#include <QString>

#include <QJsonObject>

struct **Contact**

{

QString **name**;

QString **address**;

QString **email**;

QString **picturePath**;

bool operator**==**(const Contact &**other**) const

{

return name == other.name && address == other.address && email == other.email && picturePath == other.picturePath;

}

void **toJson**(QJsonObject &**json**) const

{

json["name"] = name;

json["address"] = address;

json["email"] = email;

json["picture path"] = picturePath;

}

};

class **TableModel** : public QAbstractTableModel

{

Q\_OBJECT

**public**:

TableModel(QObject \*parent = nullptr);

**TableModel**(const List<Contact> &**contacts**, QObject \***parent** = nullptr);

int **rowCount**(const QModelIndex &**parent**) const override;

int **columnCount**(const QModelIndex &**parent**) const override;

QVariant **data**(const QModelIndex &index, int role) const override;

QVariant **headerData**(int section, Qt::Orientation orientation, int role) const override;

Qt::ItemFlags **flags**(const QModelIndex &index) const override;

bool **setData**(const QModelIndex &**index**, const QVariant &**value**, int **role** = Qt::EditRole) override;

bool **insertRows**(int **position**, int **rows**, const QModelIndex &**index** = QModelIndex()) override;

bool **removeRows**(int **position**, int **rows**, const QModelIndex &**index** = QModelIndex()) override;

const List<Contact> &**getContacts**() const;

private:

List<Contact> **contacts**;

};

#endif // TABLEMODEL\_H

\\ файл “"tablemodel.cpp”

#include "tablemodel.h"

#include "errordialog.h"

//! [0]

TableModel::**TableModel**(QObject \***parent**)

: QAbstractTableModel(parent),

contacts(List<Contact>())

{

}

TableModel::**TableModel**(const List<Contact> &**contacts**, QObject \***parent**)

: QAbstractTableModel(parent), contacts(contacts)

{

}

int TableModel::**rowCount**(const QModelIndex &**parent**) const

{

return parent.isValid() ? 0 : contacts.Count();

}

int TableModel::**columnCount**(const QModelIndex &**parent**) const

{

return parent.isValid() ? 0 : 4;

}

QVariant TableModel::**data**(const QModelIndex &index, int role) const

{

try {

if (!index.isValid())

throw IndexException("Invalid index while accesing data\n");

if (index.row() >= contacts.Count() || index.row() < 0)

throw IndexException("Invalid row in index\n");

} catch (IndexException **ex**) {

qDebug() << "Error message: " << ex.what();

qDebug() << "Error code: " << ex.GetErrorCode();

ErrorDialog **errorDialog**(nullptr, ex);

errorDialog.exec();

return QVariant();

}

const auto &**contact** = contacts.GetElement(index.row());

QString **filename** = contact.picturePath;

QPixmap **pixmap** ;

pixmap.load(filename);

switch (index.column()) {

case 0:

if (role == Qt::DisplayRole) return contact.name;

break;

case 1:

if (role == Qt::DisplayRole) return contact.address;

break;

case 2:

if (role == Qt::DisplayRole) return contact.email;

break;

case 3:

if(role == Qt::DisplayRole )

{

return filename;

}

if (role == Qt::ToolTipRole)

{

// find path for specified index

return QString("<img src='%1'>").arg(filename);

}

break;

default:

break;

}

return QVariant();

}

QVariant TableModel::**headerData**(int section, Qt::Orientation orientation, int role) const

{

if (role != Qt::DisplayRole)

return QVariant();

if (orientation == Qt::Horizontal) {

switch (section) {

case 0:

return tr("Name");

case 1:

return tr("Address");

case 2:

return tr("Email");

case 3:

return tr("Photo");

default:

break;

}

}

return QVariant();

}

bool TableModel:: **insertRows**(int **position**, int **rows**, const QModelIndex &**index**)

{

Q\_UNUSED(index);

beginInsertRows(QModelIndex(), position, position + rows - 1);

for (int row = 0; row < rows; ++row)

contacts.Insert({ QString(), QString() , QString(), QString()}, position);

endInsertRows();

return true;

}

bool TableModel::**removeRows**(int **position**, int **rows**, const QModelIndex &**index**)

{

Q\_UNUSED(index);

beginRemoveRows(QModelIndex(), position, position + rows - 1);

for (int row = 0; row < rows; ++row)

contacts.Del(position);

endRemoveRows();

return true;

}

bool TableModel::**setData**(const QModelIndex &**index**, const QVariant &**value**, int **role**)

{

try {

if(!index.isValid())

throw IndexException("Invalid index while accesing data\n");

if(value.toString() == QString())

throw ArgumentException("each value of contact must not be zero\n");

if (role == Qt::EditRole) {

const int **row** = index.row();

auto **contact** = contacts.GetElement(row);

switch (index.column()) {

case 0:

contact.name = value.toString();

break;

case 1:

contact.address = value.toString();

break;

case 2:

contact.email = value.toString();

break;

case 3:

contact.picturePath = value.toString();

break;

default:

return false;

}

contacts.SetElement(contact, row);

emit **dataChanged**(index, index, {Qt::DisplayRole, Qt::EditRole, Qt::EditRole, Qt::EditRole});

return true;

}

} catch (ArgumentException **ex**) {

qDebug() << "Error message: " << ex.what();

qDebug() << "Error code: " << ex.GetErrorCode();

ErrorDialog **errorDialog**(nullptr, ex);

errorDialog.exec();

}

catch (IndexException **ex**)

{

qDebug() << "Error message: " << ex.what();

qDebug() << "Error code: " << ex.GetErrorCode();

ErrorDialog **errorDialog**(nullptr, ex);

errorDialog.exec();

}

return false;

}

Qt::ItemFlags TableModel::**flags**(const QModelIndex &index) const

{

if (!index.isValid())

return Qt::ItemIsEnabled;

return QAbstractTableModel::flags(index) | Qt::ItemIsEditable;

}

const List<Contact> &TableModel::**getContacts**() const

{

return contacts;

}

\\ файл “"exception.h”

#ifndef EXCEPTION\_H

#define **EXCEPTION\_H**

#include <QString>

#include <QDebug>

class **Exception**

{

public:

**Exception**(QString **message**)

{

this->message = message;

}

QString **what**()

{

return message;

}

QString **GetErrorCode**()

{

return errorCode;

}

void **SetErrorCode**(QString **errorCode**)

{

this->errorCode = errorCode;

}

private:

QString **errorCode** = "C0000";

QString **message**;

};

class **ArgumentException** : public Exception

{

private:

QString **errorCode** = "C0001";

public:

**ArgumentException**(QString **msg**) : Exception(msg)

{

Exception::SetErrorCode(errorCode);

}

};

class **MemoryException** : public Exception

{

private:

QString **errorCode** = "C0002";

public:

**MemoryException**(QString **msg**) : Exception(msg)

{

Exception::SetErrorCode(errorCode);

}

};

class **IndexException** : public Exception

{

private:

QString **errorCode** = "C0003";

public:

**IndexException**(QString **msg**) : Exception(msg)

{

Exception::SetErrorCode(errorCode);

}

};

class **EmptyListException** : public Exception

{

private:

QString **errorCode** = "C0004";

public:

**EmptyListException**(QString **msg**) : Exception(msg)

{

Exception::SetErrorCode(errorCode);

}

};

#endif // EXCEPTION\_H

\\ файл “"list.h”

#ifndef LIST\_H

#define **LIST\_H**

template <class **T**>

class **Iterator**;

template <class **T**>

struct **Element**

{

T **data**; // данные

Element<T>\* **next**; // адрес следующего элемента в списке

Element<T>\* **prev**; // адрес предыдущего элемента в списке

};

template <class **T**>

class **List**

{

public:

**List**();

**List**(const List& **obj**);

void **Clear**();

void **Del**(int **index**);

void **AddEnd**(T **\_data**);

void **AddBegin**(T **\_data**);

void **Insert**(T **\_data**, int **index**);

void **Reverse**();

int **Count**() const;

void **Print**() const;

bool **isEmpty**() const;

bool **Contains**(T **\_data**) const;

T **GetElement**(int **index**) const;

void **SetElement**(T **\_data**, int **index**);

List<T>& operator**=**(const List& **obj**);

List<T>& operator**+**(const List<T>& **obj**);

bool operator**==**(const List<T>& **obj**);

bool operator**!=**(const List<T>& **obj**);

bool operator**>=**(const List<T>& **obj**);

bool operator**<=**(const List<T>& **obj**);

bool operator**>**(const List<T>& **obj**);

bool operator**<**(const List<T>& **obj**);

T operator**[]**(int **index**);

~List();

friend class Iterator<T>;

private:

void **Copy**(const List<T>& **obj**);

Element<T>\* **Move**(int **index**) const;

bool **CorrectIndex**(int **index**) const;

private:

Element<T>\* **begin**; // указатель на первый элемент списка

Element<T>\* **end**; // указатель на последний элемент списка

int **count**; // количество элементов в списке

};

template <class **T**>

class **List**;

template <class **T**>

class **Iterator**

{

public:

**Iterator**(List<T>&);

~Iterator() = default;

bool **hasNext**();

bool **hasPrevious**();

Iterator<T>& **next**();

Iterator<T>& **previous**();

Iterator<T>& operator**++**();

Iterator<T>& operator**++**(int);

Iterator<T>& operator**--**();

Iterator<T>& operator**--**(int);

Iterator<T>& operator**()**(List<T>& **list**);

T& operator**\***();

bool operator**==**(const Iterator& **other**);

bool operator**!=**(const Iterator& **other**);

bool operator**>**(const Iterator& **other**);

bool operator**<**(const Iterator& **other**);

bool operator**>=**(const Iterator& **other**);

bool operator**<=**(const Iterator& **other**);

private:

Element<T>\* **node**;

};

template<class **T**>

Iterator<T>& Iterator<T>::**next**()

{

if (hasNext())

{

this->node = this->node->next;

}

return \*this;

}

template<class **T**>

Iterator<T>& Iterator<T>::**previous**()

{

if (hasPrevious())

{

this->node = this->node->prev;

}

return \*this;

}

template<class **T**>

bool Iterator<T>::**hasNext**()

{

if (this->node != nullptr)

return true;

return false;

}

template<class **T**>

bool Iterator<T>::**hasPrevious**()

{

if (this->node != nullptr)

return true;

return false;

}

template<class **T**>

Iterator<T>::**Iterator**(List<T>& **list**)

{

node = list.begin;

}

template <class **T**>

Iterator<T>& Iterator<T>::operator**++**()

{

if (hasNext())

{

this->node = this->node->next;

}

return \*this;

}

template <class **T**>

Iterator<T>& Iterator<T>::operator**++**(int)

{

if (hasNext())

{

this->node = this->node->next;

}

return \*this;

}

template <class **T**>

Iterator<T>& Iterator<T>::operator**--**()

{

if (hasPrevious())

{

this->node = this->node->prev;

}

return \*this;

}

template <class **T**>

Iterator<T>& Iterator<T>::operator**--**(int)

{

if (hasPrevious())

{

this->node = this->node->prev;

}

return \*this;

}

template <class **T**>

Iterator<T>& Iterator<T>::operator**()**(List<T>& **list**)

{

this->node = list.begin;

return \*this;

}

template <class **T**>

T& Iterator<T>::operator**\***()

{

return this->node->data;

}

template <class **T**>

bool Iterator<T>::operator**==**(const Iterator& **other**)

{

return this->node == other.node;

}

template <class **T**>

bool Iterator<T>::operator**!=**(const Iterator& **other**)

{

return this->node != other.node;

}

template <class **T**>

bool Iterator<T>::operator**>**(const Iterator& **other**)

{

return this->node > other.node;

}

template <class **T**>

bool Iterator<T>::operator**<**(const Iterator& **other**)

{

return this->node < other.node;

}

template <class **T**>

bool Iterator<T>::operator**>=**(const Iterator& **other**)

{

return this->node > other.node;

}

template <class **T**>

bool Iterator<T>::operator**<=**(const Iterator& **other**)

{

return this->node < other.node;

}

#endif // LIST\_H

\\ файл “"list.cpp”

#include "list.h"

#include "exception.h"

template <class **T**>

List<T>::**List**()

{

// Создать пустой список

begin = end = nullptr;

count = 0;

}

template <class **T**>

List<T>::**List**(const List& **obj**)

{

// Сделать копию из списка

Copy(obj);

}

//конструктор копирования

template <class **T**>

void List<T>::**Copy**(const List<T>& **obj**)

{

// 1. Очистить список (освободить память)

Clear();

// 2. Цикл копирования this <= obj

Element<T>\* **t** = obj.begin;

while (t != nullptr)

{

AddEnd(t->data);

t = t->next;

}

}

// Очистка списка

template <class **T**>

void List<T>::**Clear**()

{

// Нужно count раз удалить первый элемент списка

int **n** = count; // сделать копию из count - важно!

for (int **i** = 0; i < n; i++)

Del(0);

}

template <class **T**>

void List<T>::**Del**(int **index**)

{

try

{

// 1. Проверка, есть ли элементы в списке

if (count == 0)

throw EmptyListException("List is empty");

// 2. Игнор, если позиция указана неправильно

if (!CorrectIndex(index))

throw IndexException("Incorrect index");

// 3. Перейти к удаляемому элементу

Element<T>\* **item** = Move(index);

// 4. Получить предыдущий элемент

Element<T>\* **itemPrev** = item->prev;

// 5. Получить следующий элемент

Element<T>\* **itemNext** = item->next;

// 6. Проверка, удаляется ли не первый элемент списка

if ((count > 1) && (itemPrev != nullptr))

itemPrev->next = itemNext; // обойти элемент item

// 7. Проверка, удаляется ли не последний элемент списка

if ((itemNext != nullptr) && (count > 1))

itemNext->prev = itemPrev;

// 8. Если удаляется первый элемент

if (index == 0)

begin = itemNext;

// 9. Если удаляется последний элемент

if (index == count - 1)

end = itemPrev;

// 10. Удалить элемент item

delete item;

// 11. Уменьшить общее количество элементов

count--;

}

catch(EmptyListException **ex**)

{

qDebug() << "Error message: " << ex.what();

qDebug() << "Error code: " << ex.GetErrorCode();

}

catch(IndexException **ex**)

{

qDebug() << "Error message: " << ex.what();

qDebug() << "Error code: " << ex.GetErrorCode();

}

}

template <class **T**>

Element<T>\* List<T>::**Move**(int **index**) const

{

// 1. Установить указатель на начало списка

Element<T>\* **t** = begin;

// 2. Перемотать в позицию index

for (int **i** = 0; i < index; i++)

t = t->next;

// 3. Вернуть указатель

return t;

}

template <class **T**>

bool List<T>::**CorrectIndex**(int **index**) const

{

return ((index >= 0) && (index <= count));

}

template <class **T**>

void List<T>::**AddEnd**(T **\_data**)

{

try

{

// 1. Создать новый элемент с данными \_data

Element<T>\* **t** = new Element<T>;

if(!t)

throw MemoryException("Error of memory");

t->next = nullptr; // следующего элемента нет

t->prev = end; // установить предыдущий элемент

t->data = \_data; // записать данные

// 2. Заполнить поле next пока что последнего элемента

if (end != nullptr)

end->next = t;

// 3. Проверка, есть ли в списке элементы

if (count == 0)

{

// если элементов нет,

// то это одновременно и начало и конец списка

begin = end = t;

}

else

{

// если элементы в списке есть, то это конец списка

end = t;

}

// 3. Увеличить общее количество элементов

count++;

}

catch (MemoryException **ex**)

{

qDebug() << "Error message: " << ex.what();

qDebug() << "Error code: " << ex.GetErrorCode();

}

}

template <class **T**>

void List<T>::**AddBegin**(T **\_data**)

{

try

{

// 1. Создать новый элемент (новую ячейку памяти)

// и заполнить его данными

Element<T>\* **t** = new Element<T>;

if(!t)

throw MemoryException("Error of memory");

t->data = \_data; // данные

t->prev = nullptr; // предыдущего элемента нет

// следующий элемент указывает на предыдущий первый

t->next = begin;

// 2. Есть ли элементы в списке?

if (count > 0)

{

// если есть, то перенаправить предыдущее начало списка

begin->prev = t;

begin = t;

}

else

{

// если элементов нет, то начало и конец есть тем самым элементом

begin = end = t;

}

// 3. Увеличить общее количество элементов

count++;

}

catch (MemoryException **ex**)

{

qDebug() << "Error message: " << ex.what();

qDebug() << "Error code: " << ex.GetErrorCode();

}

}

template <class **T**>

void List<T>::**Insert**(T **\_data**, int **index**)

{

// 1. Проверка, корректна ли позиция

if (!CorrectIndex(index))

{

return;

}

// 2. Проверка, вставка ли в конец списка (по указателю end)

if (index == count)

{

// Добавить данные по указателю end

AddEnd(\_data);

return;

}

// 3. Проверка, вставка ли в начало списка (перед begin)

if (index == 0)

{

AddBegin(\_data);

return;

}

try

{

// 4. Получить элемент перед позицией вставки

Element<T>\* **itemPrev** = Move(index - 1);

// 5. Получить элемент в позиции вставки

Element<T>\* **item** = Move(index);

// 6. Создать новый элемент и вставить его в список

Element<T>\* **t** = new Element<T>;

if(!t)

throw MemoryException("Error of memory");

t->data = \_data;

t->next = item;

t->prev = itemPrev;

itemPrev->next = t;

item->prev = t;

// 7. Увеличить количество элементов

count++;

}

catch (MemoryException **ex**)

{

qDebug() << "Error message: " << ex.what();

qDebug() << "Error code: " << ex.GetErrorCode();

}

}

template <class **T**>

void List<T>::**Reverse**()

{

List<T> **L**;

Element<T>\* **t** = begin;

// цикл формирования списка,

// элемент добавляется в начало списка

while (t != nullptr)

{

L.AddBegin(t->data);

t = t->next;

}

\*this = L;

}

template <class **T**>

int List<T>::**Count**() const

{

return count;

}

template <class **T**>

void List<T>::**Print**() const

{

Element<T>\* **t** = begin;

for (int **i** = 0; i < count; i++)

{

qDebug() << t->data << " ";

t = t->next;

}

}

template <class **T**>

bool List<T>::**isEmpty**() const

{

if(begin == nullptr && end == nullptr)

return true;

return false;

}

template<class **T**>

bool List<T>::**Contains**(T **\_data**) const

{

try

{

Element<T> \* **t** = begin;

for (int **i** = 0; i < count; ++i)

{

if(t->data == \_data)

{

return true;

}

}

return false;

}

catch(IndexException **ex**)

{

qDebug() << "Error message: " << ex.what();

qDebug() << "Error code: " << ex.GetErrorCode();

}

}

template <class **T**>

List<T>& List<T>::operator**=**(const List& **obj**)

{

Copy(obj);

return \*this;

}

template <class **T**>

T List<T>::**GetElement**(int **index**) const

{

try

{

// Проверка, корректен ли индекс,

// если индекс не корректен, сгенерировать исключение

if (!CorrectIndex(index))

throw IndexException("Incorrect index");

// Если индекс корректен, то вернуть элемент

Element<T>\* **t** = Move(index);

return t->data;

}

catch(IndexException **ex**)

{

qDebug() << "Error message: " << ex.what();

qDebug() << "Error code: " << ex.GetErrorCode();

}

}

template <class **T**>

void List<T>::**SetElement**(T **\_data**, int **index**)

{

try

{

// Проверка, корректна ли позиция

if (!CorrectIndex(index))

throw IndexException("Incorrect index");

// Получить элемент по позиции и изменить его значение

Element<T>\* **t** = Move(index);

t->data = \_data;

}

catch(IndexException **ex**)

{

qDebug() << "Error message: " << ex.what();

qDebug() << "Error code: " << ex.GetErrorCode();

}

}

template <class **T**>

List<T>& List<T>::operator**+**(const List<T>& **obj**)

{

// 1. Получить доступ к списку obj

Element<T>\* **t** = obj.begin;

// 2. Добавить к временному списку элементы t

while (t != nullptr)

{

AddEnd(t->data);

t = t->next;

}

// 3. Вернуть объединенный список

return \*this;

}

template <class **T**>

bool List<T>::operator**==**(const List<T>& **obj**)

{

// 1. Сначала сравнить размеры списков

if (count != obj.count)

return false;

// 2. Если размеры одинаковы, то сравнить поэлементно

Element<T>\* **t1** = begin;

Element<T>\* **t2** = obj.begin;

while (t1 != nullptr)

{

// Как только найдено хотя бы одно несовпадение, то выход с кодом false

if (t1->data != t2->data)

return false;

t1 = t1->next;

t2 = t2->next;

}

return true;

}

template <class **T**>

bool List<T>::operator**!=**(const List<T>& **obj**)

{

// Использовать оператор сравнения ==

return !(\*this == obj);

}

template <class **T**>

bool List<T>::operator**>=**(const List<T>& **obj**)

{

// 1. Сравнить количество элементов

if (count > obj.count)

return true;

// 2. Сравнить по содержанию

if (\*this == obj)

return true;

return false;

}

template <class **T**>

bool List<T>::operator**<=**(const List<T>& **obj**)

{

// 1. Сравнение по количеству элементов в списке

if (count < obj.count)

return true;

// 2. Сравнение по содержанию

if (\*this == obj)

return true;

return false;

}

template <class **T**>

bool List<T>::operator**>**(const List<T>& **obj**)

{

if (count > obj.count)

return true;

return false;

}

template <class **T**>

bool List<T>::operator**<**(const List<T>& **obj**)

{

if (count < obj.count)

return true;

return false;

}

template <class **T**>

T List<T>::operator**[]**(int **index**)

{

Element<T>\* **t** = Move(index);

return t->data;

}

template <class **T**>

List<T>::~List()

{

Clear(); // очистить список

}

\\ файл “"adddialog.h”

#ifndef ADDDIALOG\_H

#define **ADDDIALOG\_H**

#include <QDialog>

QT\_BEGIN\_NAMESPACE

class QLabel;

class **QPushButton**;

class **QTextEdit**;

class **QLineEdit**;

QT\_END\_NAMESPACE

class AddDialog : public QDialog

{

Q\_OBJECT

public:

AddDialog(QWidget \*parent = nullptr);

QString name() const;

QString address() const;

QString email() const;

QString getPicturePath() const;

void editAddress(const QString &name, const QString &address, const QString &email, const QString &picturePath);

private:

QLineEdit \*nameText;

QLineEdit \*addressText;

QLineEdit \*emailText;

QString picturePath;

QPushButton \*pictureButton;

private slots:

void addPicture();

};

#endif // ADDDIALOG\_H

\\ файл “"adddialog.cpp”

#include "adddialog.h"

#include <QtWidgets>

AddDialog::**AddDialog**(QWidget \***parent**)

: QDialog(parent),

nameText(new QLineEdit),

addressText(new QLineEdit),

emailText(new QLineEdit)

{

auto **nameLabel** = new QLabel(tr("Name"));

auto **addressLabel** = new QLabel(tr("Address"));

auto **emailLabel** = new QLabel(tr("Email"));

auto **pictureButton** = new QPushButton(tr("Choose Picture"));

auto **okButton** = new QPushButton(tr("OK"));

auto **cancelButton** = new QPushButton(tr("Cancel"));

auto **gLayout** = new QGridLayout;

gLayout->setColumnStretch(1, 2);

gLayout->addWidget(nameLabel, 0, 0);

gLayout->addWidget(nameText, 0, 1);

gLayout->addWidget(addressLabel, 1, 0);

gLayout->addWidget(addressText, 1, 1);

gLayout->addWidget(emailLabel, 2, 0);

gLayout->addWidget(emailText, 2, 1);

gLayout->addWidget(pictureButton, 3, 0);

auto **buttonLayout** = new QHBoxLayout;

buttonLayout->addWidget(okButton);

buttonLayout->addWidget(cancelButton);

gLayout->addLayout(buttonLayout, 3, 1, Qt::AlignRight);

auto **mainLayout** = new QVBoxLayout;

mainLayout->addLayout(gLayout);

setLayout(mainLayout);

connect(okButton, &QAbstractButton::clicked, this, &QDialog::accept);

connect(cancelButton, &QAbstractButton::clicked, this, &QDialog::reject);

connect(pictureButton ,&QAbstractButton::clicked, this, &AddDialog::addPicture);

setWindowTitle(tr("Add a Contact"));

}

QString AddDialog::**name**() const

{

return nameText->text();

}

QString AddDialog::**address**() const

{

return addressText->text();

}

QString AddDialog::**email**() const

{

return emailText->text();

}

QString AddDialog::**getPicturePath**() const

{

return picturePath;

}

void AddDialog:: **editAddress**(const QString &**name**, const QString &**address**, const QString &**email**, const QString &**picturePath**)

{

nameText->setReadOnly(true);

nameText->setText(name);

addressText->setText(address);

emailText->setText(email);

}

void AddDialog::**addPicture**()

{

QString **fileName** = QFileDialog::getSaveFileName(this, tr("Choose picture"), "/", tr("Images (\*.png \*.xpm \*.jpg)"), nullptr, QFileDialog::DontConfirmOverwrite);

if (!fileName.isEmpty())

{

picturePath = fileName;

}

}

\\ файл “"errordialog.h”

#ifndef ERRORDIALOG\_H

#define **ERRORDIALOG\_H**

#include "exception.h"

#include <QDialog>

#include <QWidget>

QT\_BEGIN\_NAMESPACE

class QLabel;

QT\_END\_NAMESPACE

class ErrorDialog : public QDialog

{

Q\_OBJECT

public:

ErrorDialog(QWidget \* parent, Exception ex);

};

#endif // ERRORDIALOG\_H

\\ файл “"errordialog.cpp”

#include "errordialog.h"

#include <QtWidgets>

ErrorDialog::**ErrorDialog**(QWidget \* **parent**, Exception **ex**)

:QDialog(parent)

{

setWindowTitle(tr("Error!"));

QLabel \***errorLabel** = new QLabel(tr("Error message : ") + ex.what(), this);

QLabel \***codeLabel** = new QLabel(tr("Error code : ") + ex.GetErrorCode(), this);

auto \***mainLayout** = new QVBoxLayout(this);

mainLayout->addWidget(errorLabel);

mainLayout->addWidget(codeLabel);

setLayout(mainLayout);

}

\\ файл “"finddialog.h”

#ifndef FINDDIALOG\_H

#define **FINDDIALOG\_H**

#include <QDialog>

QT\_BEGIN\_NAMESPACE

class QLabel;

class **QPushButton**;

class **QTextEdit**;

class **QLineEdit**;

QT\_END\_NAMESPACE

class FindDialog : public QDialog

{

Q\_OBJECT

public:

FindDialog(QWidget \*parent = nullptr);

QString name() const;

QString address() const;

QString email() const;

private:

QLineEdit \*nameText;

QLineEdit \*addressText;

QLineEdit \*emailText;

};

#endif // FINDDIALOG\_H

\\ файл “"finddialog.cpp”

#include "finddialog.h"

#include <QtWidgets>

FindDialog::**FindDialog**(QWidget \***parent**)

: QDialog(parent),

nameText(new QLineEdit),

addressText(new QLineEdit),

emailText(new QLineEdit)

{

auto **nameLabel** = new QLabel(tr("Name"));

auto **addressLabel** = new QLabel(tr("Address"));

auto **emailLabel** = new QLabel(tr("Email"));

auto **okButton** = new QPushButton(tr("OK"));

auto **cancelButton** = new QPushButton(tr("Cancel"));

auto **gLayout** = new QGridLayout;

gLayout->setColumnStretch(1, 2);

gLayout->addWidget(nameLabel, 0, 0);

gLayout->addWidget(nameText, 0, 1);

gLayout->addWidget(addressLabel, 1, 0);

gLayout->addWidget(addressText, 1, 1);

gLayout->addWidget(emailLabel, 2, 0);

gLayout->addWidget(emailText, 2, 1);

auto **buttonLayout** = new QHBoxLayout;

buttonLayout->addWidget(okButton);

buttonLayout->addWidget(cancelButton);

gLayout->addLayout(buttonLayout, 3, 1, Qt::AlignRight);

auto **mainLayout** = new QVBoxLayout;

mainLayout->addLayout(gLayout);

setLayout(mainLayout);

connect(okButton, &QAbstractButton::clicked, this, &QDialog::accept);

connect(cancelButton, &QAbstractButton::clicked, this, &QDialog::reject);

setWindowTitle(tr("Find a Contact"));

}

QString FindDialog::**name**() const

{

return nameText->text();

}

QString FindDialog::**address**() const

{

return addressText->text();

}

QString FindDialog::**email**() const

{

return emailText->text();

}

# ПРИЛОЖЕНИЕ Б

*(обязательное)*

UML-диаграмма классов

# ПРИЛОЖЕНИЕ В

*(обязательное)*

Блок-схемы алгоритмов функций void AddressWidget::addEntry(const QString &name, const QString &address, const QString &email, const QString &picturePath), void AddressWidget::findEntry(), void AddressWidget::writeToFile(const QString &fileName), void AddressWidget::readFromFile(const QString &fileName), bool TableModel::setData(const QModelIndex &index, const QVariant &value, int role)

# ПРИЛОЖЕНИЕ Г

*(обязательное)*

Ведомость документов.