

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ  
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. ПЕТРА ВЕЛИКОГО

Институт Компьютерных наук и технологий

Высшая школа искусственного интеллекта

Направление 02.03.01 Математика и компьютерные науки

Отчет по курсовой работе

Реализация картотеки в виде таблицы на языке C++

Группа: 3530201/10002

Студент: \_\_\_\_\_

Салимли А.М.

Преподаватель: \_\_\_\_\_

Глазунов Вадим Валерьевич

«\_\_\_\_\_» \_\_\_\_\_ 20\_\_г.

Санкт-Петербург – 2022

# Содержание

<b>Введение</b>	<b>3</b>
<b>1 Постановка задачи</b>	<b>4</b>
<b>2 Реализация</b>	<b>5</b>
2.1 Библиотечные классы . . . . .	5
2.2 Пользовательские классы . . . . .	6
2.2.1 MyWidget . . . . .	6
2.2.2 MyTerran . . . . .	8
2.2.3 AddNew . . . . .	9
2.2.4 EditLine . . . . .	11
<b>3 Тестирование</b>	<b>14</b>
<b>Заключение</b>	<b>21</b>
<b>Источники</b>	<b>22</b>
<b>Приложение</b>	<b>23</b>
main.cpp . . . . .	23
MyWidget.h . . . . .	23
MyWidget.cpp . . . . .	24
MyTerran.h . . . . .	31
MyTerran.cpp . . . . .	32
AddNew.h . . . . .	33
AddNew.cpp . . . . .	35
EditLine.h . . . . .	38
EditLine.cpp . . . . .	39
GlobVar.h . . . . .	43

## Введение

В курсовой работе требуется реализовать таблицу-картотеку, в которой хранятся данные о людях. Их ФИО, домашний адрес, номер телефона и адрес электронной почты.

Задание нужно выполнить при помощи Qt. Qt — фреймворк для разработки кроссплатформенного программного обеспечения на языке программирования C++. Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML. Является полностью объектно-ориентированным, расширяемым и поддерживающим технику компонентного программирования.

# 1 Постановка задачи

Нужно реализовать картотеку в виде таблицы. Для этого требуется:

- Реализовать таблицу, хранящую данные при помощи графической библиотеки Qt
- Реализовать хранение данных
- Реализовать добавление данных из файла и сохранение данных в файл
- Реализовать добавление пользователем в таблицу данных о людях
- Реализовать проверку вводимых пользователем данных при помощи регулярных выражений. Проверять поля фамилии, имени, отчества, даты рождения, e-mail'a и номера телефона.
- Реализовать редактирование данных в таблице
- Реализовать удаление данных из таблицы
- Реализовать поиск строк в таблице, содержащих в любой из ячеек введенное пользователем слово-идентификатор. Совпадение должно быть полным.

Пояснение к проверке полей:

Для фамилии, имени и отчества происходит проверка такая, чтобы первая буква была заглавной, остальные прописными. Допускается использование дефиса, но он не может быть первым или последним.

Для даты рождения происходит проверка такая, чтобы дата была не больше текущей даты, а так же не меньше, чем дата рождения самого старого человека на земле на момент написания отчета.

Для адреса электронной почты происходит проверка такая, чтобы адрес состоял только из латинских символов, в нем обязательно присутствовал символ '@', но символ '@' не может быть последним. Также обязательно разделение доменной части на 2 уровня при помощи символа '.'.

Для номера телефона происходит проверка такая, чтобы вводимый телефон соответствовал формату:

'+'КодСтраны(КодРегиона)ТриЦифры'-ДвеЦифры'-ДвеЦифры.

## 2 Реализация

### 2.1 Библиотечные классы

- QApplication - руководит управляющей логикой GUI и основными настройками.
- QWidget - базовый класс для всех объектов интерфейса.
- QGridLayout - располагает виджеты по сетке.
- QPushButton - позволяет создавать кнопки.
- QFile - дает возможность работать с файлами, загружать из файлов и сохранять в файлы.
- QLabel - дает возможность отражать в окне приложения текст.
- QTableWidgetItem - предоставляет таблицу для просмотра элементов и взаимодействия с ними.
- QHeaderView - дает возможность использовать строки заголовков и столбцов как элементы.
- QDate - предоставляет функции для работы с датами.
- QDateEdit - предоставляет виджет для редактирования и изменения дат на основе виджета QDateTimeEdit.
- QLineEdit - представляет собой однострочный текстовый редактор.
- QList - класс шаблонов, дающий возможность работать с контейнером list.
- QTextStream - предоставляет удобный интерфейс для чтения и записи текста.
- QString - класс, дающий возможность работать с типом string.
- QStringList - предоставляет возможность работать со списками строк.
- QDialog - базовый класс для реализации диалоговых окон.
- QRegExp - дает возможность сопоставлять введенную пользователем строку с шаблоном, который реализуется при помощи регулярных выражений.
- QRegExpValidator - нужен для проверки введенной строки на соответствие регулярному выражению.
- QMessageBox - дает возможность использовать модальное диалоговое окно для информирования пользователя или для того, чтобы задать пользователю вопрос и получить ответ.

## 2.2 Пользовательские классы

Пользовательские классы связаны между собой. MyWidget является основным, в нем используются три других пользовательских класса. В классах AddNew и EditLine происходит создание/редактирование объектов типа MyTerran.

### 2.2.1 MyWidget

Класс наследуется от QWidget. Реализует окно, в котором отображается таблица и кнопки.

#### Поля:

- `QGridLayout* grid;`
- `QTableWidget* table;`
- `QPushButton* btnAdd;`
- `QPushButton* btnEdit;`
- `QPushButton* btnDelete;`
- `QPushButton* btnLoad;`
- `QPushButton* btnSave;`
- `QPushButton* btnExit;`
- `QLineEdit* LiEdSearch;` - строка, в которую водится ключевое слово для поиска.
- `QPushButton* btnSearch;`
- `QPushButton* btnNext;` - кнопка для переключения на следующий найденный элемент.
- `QPushButton* btnPrev;` - кнопка для переключения на предыдущий найденный элемент.
- `QList<MyTerran> terran;` - список всех хранящихся в таблице объектов.
- `QList<QTableWidgetItem*> found;` - список для хранения указателей всех найденных во время поиска объектов.
- `QList<QTableWidgetItem*>::iterator iterFoun;` - итератор для движения в цикле по `QList<QTableWidgetItem*> found`.

#### Методы:

- `MyWidget::MyWidget(QWidget* parent) : QWidget(parent)` - конструктор, в котором инициализируются поля, формируется и настраивается таблица, таблица и кнопки располагаются по сетке, связываются сигналы и слоты.
- `void MyWidget::FillRow(int index)` - метод, при помощи которого поля строки таблицы заполняются значениями полей объектов класса `MyTerrain` из `QList`. В качестве параметра принимает номер строки, которую надо заполнить.
- `void MyWidget::AddClicked()` - слот, вызываемый при нажатии кнопки добавления новой строки в таблицу. Вызывает диалоговое окно, реализованное классом `AddNew`, и, если возвращенный из окна объект `MyTerrain` не пустой, то вызывает метод `AddPerson(MyTerrain p)`, описанный следующим. После добавления перезапускается сортировка по колонке, если таковая имелаась. Делается это для корректного отображения данных в таблице.
- `void MyWidget::AddPerson(MyTerrain p)` - метод, добавляющий в `QList terrain` новый объект. Так же вызывается метод `FillRow(terrain.size() - 1)` для отображения добавленного элемента в таблице.
- `void MyWidget::EditClicked()` - слот, вызываемый при нажатии кнопки редактирования строки. Вызывает диалоговое окно, реализованное классом `EditLine`, и, если возвращенный из окна объект `MyTerrain` не пустой, то заменяет выбранный объект отредактированной версией. После редактирования перезапускается сортировка по колонке, если таковая имелаась. Делается это для корректного отображения данных в таблице.
- `void MyWidget::LoadClicked()` - слот, вызываемый при нажатии кнопки загрузки элементов в таблицу из файла. Добавляет элементы в таблицу при помощи метода `AddPerson`. После добавления перезапускается сортировка по колонке, если таковая имелаась.
- `void MyWidget::SaveClicked()` - слот, вызываемый при нажатии на кнопку сохранения таблицы в файл. Сохраняет значения таблицы в текстовый файл «output.txt».
- `void MyWidget::DelClicked()` - слот, вызываемый при нажатии кнопки удаления элемента из таблицы. Удаляет элемент из `QList terrain` и удаляет строку из таблицы.
- `void MyWidget::UpdAfterSort(int col, Qt::SortOrder order)` - слот, вызываемый при сортировке таблицы по колонкам. Сортирует `QList terrain` в соответствии с порядком сортировки, выбранным пользователем. Нужен для того, чтобы при сортировке таблицы изменялась не только визуальное отображение порядка элементов, но и их порядок в `QList terrain`. Сортировка происходит при

помощи вспомогательного класса `SortFunctor`, а точнее при помощи перегруженного во вспомогательном классе предиката `'()`'.

- `void MyWidget::LEUpdate(const QString& text)` - слот, отвечающий за отображение кнопок поиска, перехода на следующий найденный и предыдущий. Если строка поиска пуста, кнопки не активны.
- `void MyWidget::SearchClicked()` - слот, вызываемый при нажатии на кнопку поиска. Выделяет строку, в которой находится найденный элемент.
- `void MyWidget::NextClicked()` - слот, вызываемый при нажатии кнопки следующий при активном поиске. Выделяет следующую строку, в которой находится подходящий элемент, если такая имеется.
- `void MyWidget::PrevClicked()` - слот, вызываемый при нажатии кнопки предыдущий при активном поиске. Выделяет предыдущую строку, в которой находится подходящий элемент, если такая имеется.

### 2.2.2 MyTerran

Класс ни от чего не наследуется. Реализует хранение данных о человеке, его ФИО, дату рождения, адрес, email, телефон.

#### Поля:

- `QString TerLaN;` - хранит фамилию.
- `QString TerFiN;` - хранит имя.
- `QString TerSeN;` - хранит отчество.
- `QString TerBirth;` - хранит дату рождения.
- `QString TerAddres;` - хранит домашний адрес.
- `QString TerEmail;` - хранит адрес электронной почты.
- `QString TerPhone;` - хранит номер телефона.

#### Методы:

- `MyTerran::MyTerran(QString lastName, QString firstName, QString secName, QString birth, QString address, QString email, QString phone)` - конструктор, принимающий в качестве параметров объекты типа `QString` и присваивающий их значения соответствующим полям. По умолчанию инициализирует все параметры кроме даты рождения равны `' '`, дата рождения равна «02.01.1903».



- `MyTerran::MyTerran(QStringList list)` - конструктор, принимающий в качестве параметра список `QStringList`, полями которого инициализируются поля объекта типа `MyTerran`.
- `QString MyTerran::GetLastName()` - метод, возвращающий значение поля, хранящего фамилию.
- `QString MyTerran::GetFirstName()` - метод, возвращающий значение поля, хранящего имя.
- `QString MyTerran::GetSecName()` - метод, возвращающий значение поля, хранящего отчество.
- `QString MyTerran::GetBirth()` - метод, возвращающий значение поля, хранящего дату рождения.
- `QString MyTerran::GetAddress()` - метод, возвращающий значение поля, хранящего домашний адрес.
- `QString MyTerran::GetEmail()` - метод, возвращающий значение поля, хранящего email.
- `QString MyTerran::GetPhone()` - метод, возвращающий значение поля, хранящего телефон.
- `QStringList MyTerran::GetAll()` - метод, возвращающий `QStringList`, в который записаны значения всех полей.
- `bool MyTerran::operator==(const MyTerran& p) const` - перегруженный оператор '=='.
- `friend QTextStream& operator<<(QTextStream& s, const MyTerran& p)` - перегруженный оператор '«' для вывода в текстовый поток значения всех полей через запятую.

### 2.2.3 AddNew

Класс наследуется от `QDialog`. Реализует диалоговое окно добавления новой строки в таблицу.

**Поля:**

- `QGridLayout* grid;`
- `QLabel* lblLN;`
- `QLabel* lblFN;`

- QLabel\* lblSN;
- QLabel\* lblBirth;
- QLabel\* lblAddress;
- QLabel\* lblEmail;
- QLabel\* lblPhone;
- QLabel\* temaplatePhone;
- QLineEdit\* LiEdLN; - строка для ввода фамилии.
- QLineEdit\* LiEdFN; - строка для ввода имени.
- QLineEdit\* LiEdSN; - строка для ввода отчества.
- QLineEdit\* LiEdAddress; - строка для ввода адреса.
- QLineEdit\* LiEdEmail; - строка для ввода адреса электронной почты.
- QLineEdit\* LiEdPhone; - строка для ввода номера телефона.
- QDateEdit\* DaEdBirth; - элемент интерфейса для изменения даты рождения.
- QPushButton\* btnOk;
- QPushButton\* btnCancel;
- QRegExp regexName; - поле, инициализирующееся в конструкторе регулярным выражением для проверки ФИО.
- QRegExp regexEmail; - поле, инициализирующееся в конструкторе регулярным выражением для проверки email'а.
- QRegExp regexPhone; - поле, инициализирующееся в конструкторе регулярным выражением для проверки номера телефона.
- MyTerrain inpPerson; - поле для хранения объекта типа MyTerrain. В конструкторе инициализируются конструктором MyTerrain по умолчанию.
- MyTerrain\* outPerson; - поле для хранения созданного объекта типа MyTerrain и последующего его занесения в таблицу в методе класса MyWidget.
- QStringList notFilled; - список строк, в который добавляются незаполненные строки и затем выводятся на экран с соответствующим предупреждением.
- QStringList notValidated; - список строк, в который добавляются некорректно заполненные строки. Выводятся на экран в отдельном окне при попытке создания объекта.

## Методы:

- `AddNew::AddNew(QString title, QWidget* parent)` - конструктор, в котором инициализируются поля, настраиваются регулярные выражения; строки для ввода, надписи и кнопки располагаются по сетке, связываются сигналы и слоты.
- `AddNew::~AddNew()` - деструктор, в котором удаляется создаваемый объект `outPerson` типа `MyTerran`.
- `MyTerran* AddNew::GetOutputPerson()` - метод, возвращающий объект `outPerson`.
- `bool AddNew::IsAllFilled()` - метод, в котором проверяется, заполнены ли все строки ввода. Если какая-то строка не заполнена, её название добавляется в список `notFilled`.
- `bool AddNew::ValidateInput()` - метод, проверяющий правильность заполнения строк ввода при помощи регулярных выражений. Если какая-то строка не прошла проверку, её название добавляется в список `notValidated`.
- `void AddNew::ThrowWarning(QString msg, QStringList& list)` - метод, создающий окно предупреждения, в котором выводится сообщение `msg`, в котором содержатся строки принимаемого в качестве параметра `QStringList& list`.
- `void AddNew::accept()` - слот, вызываемый при окончании работы с диалоговым окном. Если все строки заполнены, и заполнены корректно, то в `outPerson` создается объект типа `MyTerran` с параметрами, взятыми из строк ввода. Если какая-то строка или строки не заполнены, выведется соответствующее предупреждение. Если какая-то строка или строки заполнены не корректно, выведется соответствующее предупреждение.

### 2.2.4 EditLine

Класс наследуется от `QDialog`. Реализует диалоговое окно редактирования данных в таблице.

#### Поля:

- `QGridLayout* grid;`
- `QLabel* lblLN;`
- `QLabel* lblFN;`
- `QLabel* lblSN;`
- `QLabel* lblBirth;`

- QLabel\* lblAddress;
- QLabel\* lblEmail;
- QLabel\* lblPhone;
- QLabel\* temaplatePhone;
- QLineEdit\* LiEdLN; - строка для ввода фамилии.
- QLineEdit\* LiEdFN; - строка для ввода имени.
- QLineEdit\* LiEdSN; - строка для ввода отчества.
- QLineEdit\* LiEdAddress; - строка для ввода адреса.
- QLineEdit\* LiEdEmail; - строка для ввода адреса электронной почты.
- QLineEdit\* LiEdPhone; - строка для ввода номера телефона.
- QDateEdit\* DaEdBirth; - элемент интерфейса для изменения даты рождения.
- QPushButton\* btnOk;
- QPushButton\* btnCancel;
- QRegExp regexName; - поле, инициализирующееся в конструкторе регулярным выражением для проверки ФИО.
- QRegExp regexEmail; - поле, инициализирующееся в конструкторе регулярным выражением для проверки email'а.
- QRegExp regexPhone; - поле, инициализирующееся в конструкторе регулярным выражением для проверки номера телефона.
- MyTerran\* inpPerson; - поле, хранящее редактируемый объект типа MyTerran.
- MyTerran\* outPerson; - поле для хранения созданного объекта типа MyTerran и последующего его занесения в таблицу в методе класса MyWidget.
- QStringList notFilled; - список строк, в который добавляются незаполненные строки и затем выводятся на экран с соответствующим предупреждением.
- QStringList notValidated; - список строк, в который добавляются некорректно заполненные строки. Выводятся на экран в отдельном окне при попытке создания объекта.

## Методы:

- `EditLine::EditLine(QString title, MyTerrain* p, QWidget* parent)` - конструктор, в котором инициализируются поля, настраиваются регулярные выражения; строки для ввода, надписи и кнопки располагаются по сетке, связываются сигналы и слоты.
- `EditLine::~EditLine()` - деструктор, в котором удаляется созданный после редактирования `inpPerson` объект `outPerson`.
- `MyTerrain* EditLine::GetOutputPerson()` - метод, возвращающий объект `outPerson`.
- `bool EditLine::IsAllFilled()` - метод, в котором проверяется, заполнены ли все строки ввода. Если какая-то строка не заполнена, её название добавляется в список `notFilled`.
- `bool EditLine::IsEdited()` - метод, возвращающий `true`, если `outPerson`, полученный в результате редактирования `inpPerson`, отличается от `inpPerson`.
- `bool EditLine::ValidateInput()` - метод, проверяющий правильность заполнения строк ввода при помощи регулярных выражений. Если какая-то строка не прошла проверку, её название добавляется в список `notValidated`.
- `void EditLine::ThrowWarning(QString msg, QStringList& list)` - метод, создающий окно предупреждения, в котором выводится сообщение `msg`, в котором содержатся строки принимаемого в качестве параметра `QStringList& list`.
- `void EditLine::accept()` - слот, вызываемый при окончании работы с диалоговым окном. Если все строки заполнены, и заполнены корректно, то в `outPerson` создается объект типа `MyTerrain` с параметрами, взятыми из строк ввода. Если какая-то строка или строки не заполнены, выведется соответствующее предупреждение. Если какая-то строка или строки заполнены не корректно, выведется соответствующее предупреждение.

### 3 Тестирование

На Рис.1 изображено приложение при начале работы. Таблица пуста.

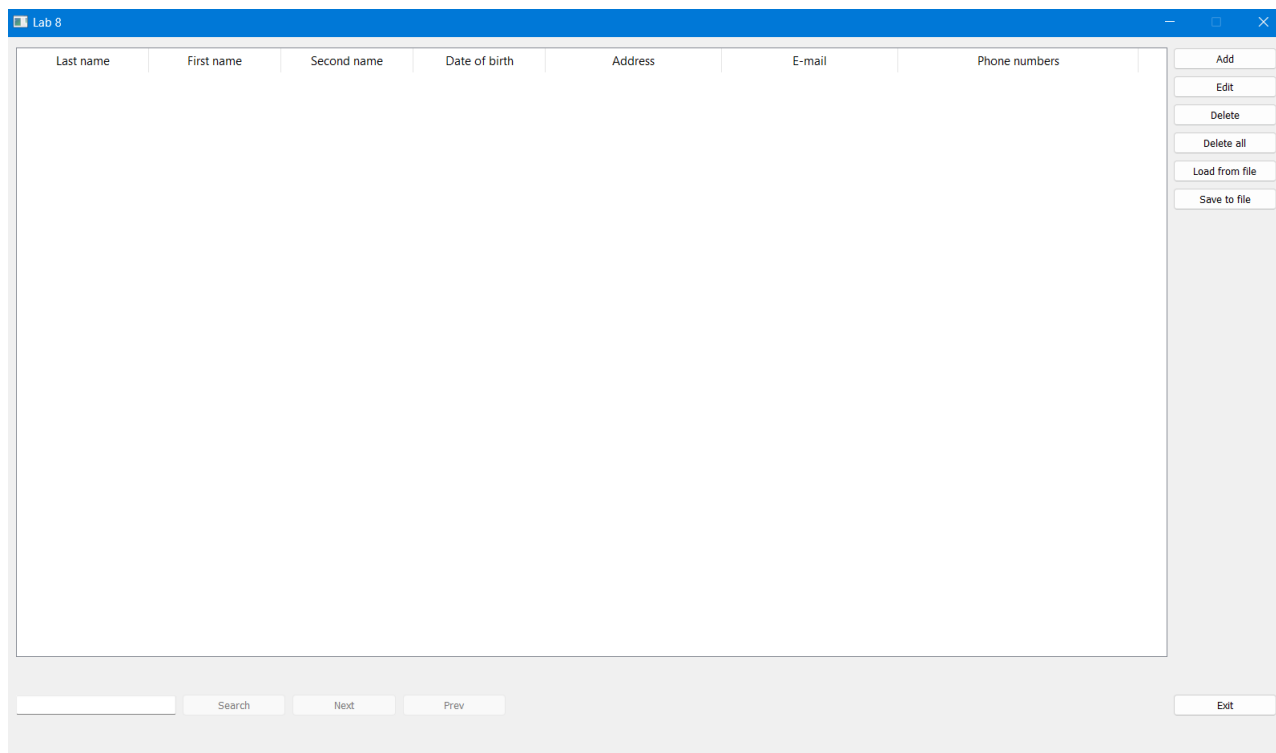


Рис.1: Начало работы

На Рис.2 изображено окно добавление новой строки в таблицу.

addwindow.jpg

Рис.2: Окно добавления

На Рис.3 изображено окно предупреждения о том, что выведенные в сообщении строки не заполнены.

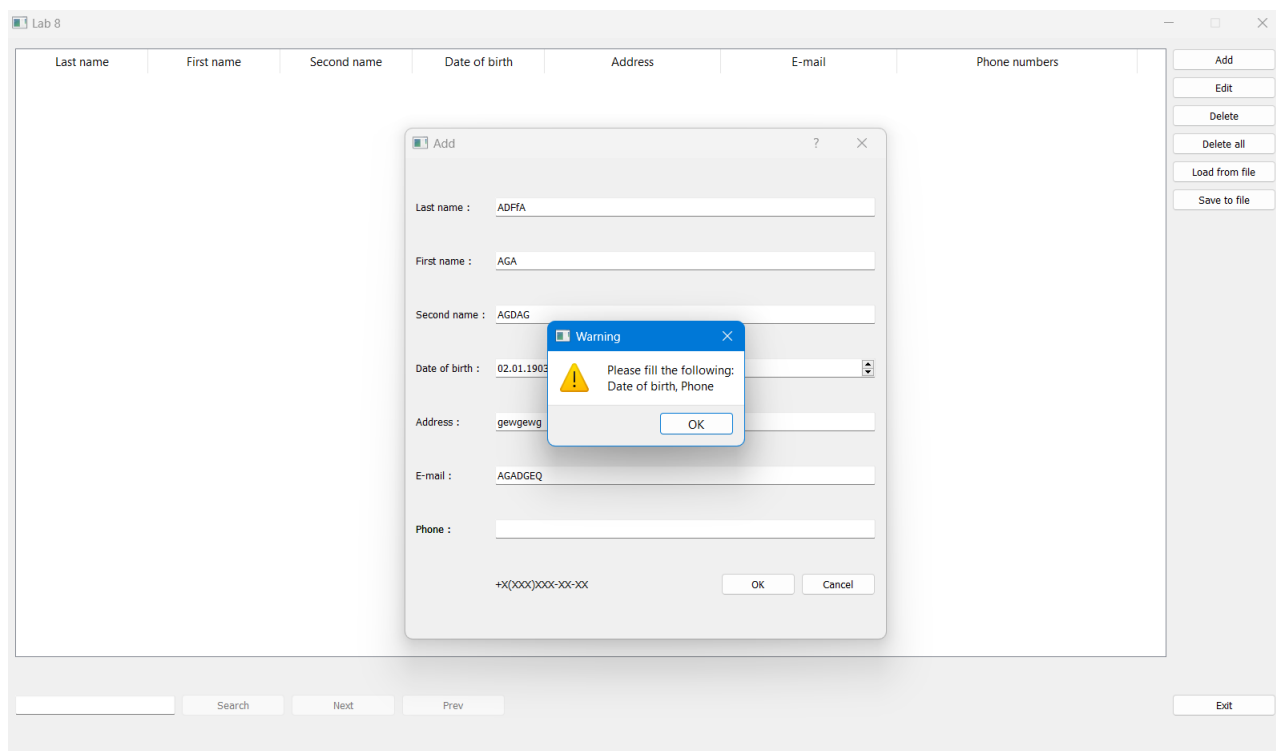


Рис.3: Окно предупреждения о заполнении

На Рис.4 изображено окно предупреждения о том, что выведенные в сообщении строки заполнены некорректно.

uncorrectfill.jpg

Рис.4: Окно предупреждения о некорректном вводе

На Рис.5 изображена таблица, содержащая несколько элементов.

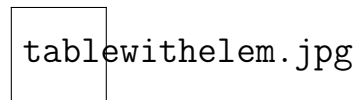


Рис.5: Заполненная таблица

На Рис.6 изображено окно редактирования элемента.

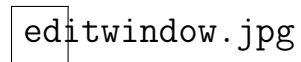


Рис.6: Окно редактирования



На Рис.7 изображена таблица с выбранной строкой.


selectedrow.jpg

Рис.7: Таблица с выбранным элементом

На Рис.8 изображена таблица из Рис.7 после сортировки по первому столбцу.


aftersort.jpg

Рис.8: Отсортированная по первому столбцу таблица

На Рис.9 изображена таблица из Рис.8 после удаления одного элемента.


 afterdelete.jpg

Рис.9: Таблица после удаления одного элемента

На Рис.10 изображена таблица из Рис.9 после загрузки трех элементов из файла.


 afterloadfromfile.jpg

Рис.10: Таблица после загрузки элементов из файла

На Рис.11 изображен файл формата .txt, в который была сохранена таблица, изображенная на Рис.10.


 savedinfile.jpg

Рис.11: Сохраненная в файл таблица

На Рис.12 изображена таблица с Рис.10, в которой был совершен поиск строки, содержащей введенное слово. Найденная строка выделена.



 search.jpg

Рис.12: Работа поиска по таблице

На Рис.13 изображена таблица с Рис.12. Была нажата кнопка «Next», из-за чего выделена следующая строка, содержащая введенное слово.



searchNextbtn.jpg

Рис.13: Выделение следующей строки, подходящей под критерий поиска.

## Заключение

В результате выполнения курсовой работы было реализовано оконное приложение «Картотека», обеспечивающее работу с таблицей, содержащей данные о людях. В ходе работы было реализовано следующее:

- Был реализован оконный интерфейс при помощи графической библиотеки Qt, класса MyWidget, класса TableWidget.
- Было реализовано хранение данных при помощи контейнера QList, хранящего объекты типа MyTerrain, содержащие конкретные данные о людях
- Было реализовано добавление данных из файла и сохранение данных в файл при помощи классов QFile и QTextStream.
- Было реализовано добавление пользователем данных в таблицу при помощи класса AddNew
- Было реализовано редактирование записей в таблице при помощи класса EditLine.
- Была реализована проверка вводимых данных на соответствие шаблону. Проверка выполняется при помощи регулярных выражений и типа QDateTime.
- Было реализовано удаление данных из таблицы при помощи метода DelClicked() класса MyWidget.
- Был реализован поиск по значению при помощи итератора и хранения данных при помощи контейнера QList.

Среда разработки: Visual Studio 2019

Версия фреймворка: Qt 5.15.2

## **Источники**

1. Qt Documentation [Электронный ресурс]:  
URL: <https://doc.qt.io> (дата обращения: 09.12.2021).
2. Форум программистов Киберфорум [Электронный ресурс]:  
URL: <https://www.cyberforum.ru> (дата обращения: 10.12.2021).

## Приложение

### main.cpp

```
#include <QtWidgets/QApplication>
#include "MyWidget.h"
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MyWidget mw;
    mw.show();
    return a.exec();
}
```

### MyWidget.h

```
#include <QApplication>
#include <QWidget>
#include <QPushButton>
#include <QGridLayout>
#include <QFile>
#include <QTableWidget>
#include <QHeaderView>
#include <QDate>
#include <QLineEdit>
#include <QList>
#include <QTextStream>
#include <algorithm>
#include "MyTerran.h"
#include "AddNew.h"
#include "EditLine.h"
#include "GlobVar.h"

class MyWidget : public QWidget {
    Q_OBJECT
private:
    QGridLayout* grid;
    QTableWidget* table;
    QPushButton* btnAdd;
    QPushButton* btnEdit;
    QPushButton* btnDelete;
```

```

        QPushButton* btnLoad;
        QPushButton* btnSave;
        QPushButton* btnExit;
        QLineEdit* LiEdSearch;
        QPushButton* btnSearch;
        QPushButton* btnNext;
        QPushButton* btnPrev;
        QList<MyTerran> terran;
        QList<QTableWidgetItem*> found;
        QList<QTableWidgetItem*>::iterator iterFoun;
        void FillRow(int index);
public:
        MyWidget(QWidget* parent = Q_NULLPTR);
        void AddPerson(MyTerran person);
public slots:
        void AddClicked();
        void EditClicked();
        void LoadClicked();
        void SaveClicked();
        void DelClicked();
        void UpdAfterSort(int, Qt::SortOrder order);
        void LEUpdate(const QString& text);
        void SearchClicked();
        void NextClicked();
        void PrevClicked();
};

class SortFunctor {
private:
        int col;
        Qt::SortOrder order;
public:
        SortFunctor(int c, Qt::SortOrder o);
        bool operator()(MyTerran& p1, MyTerran& p2);
};

```

## MyWidget.cpp

```

#include "MyWidget.h"
MyWidget::MyWidget(QWidget* parent) : QWidget(parent) {
        this->setWindowTitle("Lab 8");
}

```



```

this->setMinimumSize(1600, 900);
this->setMaximumSize(1600, 900);
this->resize(1600, 900);
grid = new QGridLayout(this);
table = new QTableWidgetItem(this);
btnAdd = new QPushButton("Add", this);
btnEdit = new QPushButton("Edit", this);
btnDelete = new QPushButton("Delete", this);
btnLoad = new QPushButton("Load from file", this);
btnSave = new QPushButton("Save to file", this);
btnExit = new QPushButton("Exit", this);
grid->addWidget(table, 0, 0, 20, 10);
grid->addWidget(btnAdd, 0, 21, 1, 1);
grid->addWidget(btnEdit, 1, 21, 1, 1);
grid->addWidget(btnDelete, 2, 21, 1, 1);
grid->addWidget(btnLoad, 3, 21, 1, 1);
grid->addWidget(btnSave, 4, 21, 1, 1);
grid->addWidget(btnExit, 21, 21, 1, 1);
table->setSortingEnabled(true);
table->setEditTriggers(QAbstractItemView::NoEditTriggers);
table->setSelectionBehavior(QAbstractItemView::SelectRows);
table->setSelectionMode(QAbstractItemView::SingleSelection);
table->setColumnCount(ColumnCount);
table->setHorizontalHeaderLabels(QStringList() << "Last name"

<< "First name"          << "Second name" << "Date of birth"

<< "Address" << "E-mail" << "Phone" );
table->setColumnWidth(0, 165);
table->setColumnWidth(1, 165);
table->setColumnWidth(2, 165);
table->setColumnWidth(3, 165);
table->setColumnWidth(4, 250);
table->setColumnWidth(5, 250);
table->setColumnWidth(6, 250);
QObject::connect(
    table->horizontalHeader(),
    SIGNAL(sortIndicatorChanged(int, Qt::SortOrder)),
    this, SLOT(UpdAfterSort(int, Qt::SortOrder)));
QObject::connect(btnAdd, SIGNAL(clicked()),

```

```

        this, SLOT(AddClicked()));
QObject::connect(btnEdit, SIGNAL(clicked()),
    this, SLOT(EditClicked()));
QObject::connect(btnDelete, SIGNAL(clicked()),
    this, SLOT(DelClicked()));
QObject::connect(btnLoad, SIGNAL(clicked()),
    this, SLOT(LoadClicked()));
QObject::connect(btnLoad, SIGNAL(clicked()),
    this, SLOT(UpdAfterSort(int, Qt::SortOrder)));
QObject::connect(btnSave, SIGNAL(clicked()),
    this, SLOT(SaveClicked()));
QObject::connect(btnExit, SIGNAL(clicked()),
    qApp, SLOT(quit()));
LiEdSearch = new QLineEdit(this);
btnSearch = new QPushButton("Search", this);
btnNext = new QPushButton("Next", this);
btnPrev = new QPushButton("Prev", this);
grid->addWidget(LiEdSearch, 20, 0, 3, 1);
grid->addWidget(btnSearch, 21, 1, 1, 1);
grid->addWidget(btnNext, 21, 2, 1, 1);
grid->addWidget(btnPrev, 21, 3, 1, 1);
LiEdSearch->setMaximumWidth(200);
btnSearch->setEnabled(false);
btnNext->setEnabled(false);
btnPrev->setEnabled(false);
QObject::connect(LiEdSearch, SIGNAL(textChanged(const QString&)),
    this, SLOT(LEUpdate(const QString&)));
QObject::connect(btnSearch, SIGNAL(clicked()),
    this, SLOT(SearchClicked()));
QObject::connect(btnNext, SIGNAL(clicked()),
    this, SLOT(NextClicked()));
QObject::connect(btnPrev, SIGNAL(clicked()),
    this, SLOT(PrevClicked()));
}

void MyWidget::FillRow(int index) {
    for (int i = 0; i < ColumnCount; i++) {
        QTableWidgetItem* item = table->item(index, i);
        delete item;
        item = new QTableWidgetItem(terran[index].GetAll()[i]);
        table->setItem(index, i, item);
    }
}

```

```

    }
}

void MyWidget::AddPerson(MyTerran p) {
    table->setSortingEnabled(false);
    terran.append(p);
    table->insertRow(terran.size() - 1);
    FillRow(terran.size() - 1);
    table->setSortingEnabled(true);
}

void MyWidget::AddClicked() {
    table->setSortingEnabled(false);
    AddNew* add = new AddNew("Add", this);
    if (add->exec() == QDialog::Accepted) {
        if (add->GetOutputPerson() != nullptr) {
            AddPerson(*add->GetOutputPerson());
        }
    }
    delete add;
    int column = table->horizontalHeader()->sortIndicatorSection();
    auto ord = table->horizontalHeader()->sortIndicatorOrder();
    if (column != 7) {
        table->sortItems(column, Qt::DescendingOrder);
        table->sortItems(column, Qt::AscendingOrder);
        table->sortItems(column, ord);
    }
}

void MyWidget::EditClicked() {
    table->setSortingEnabled(false);
    QItemSelectionModel* select = table->selectionModel();
    if (select->hasSelection()) {
        int row = select->selectedRows()[0].row();
        EditLine* edit =
            new EditLine("Edit", &terran[row], this);
        if (edit->exec() == QDialog::Accepted) {
            if (edit->GetOutputPerson() != nullptr) {
                terran[row] = *edit->GetOutputPerson();
                FillRow(row);
            }
        }
        delete edit;
    }
}

```

```

    }
    table->setSortingEnabled(true);
    int column = table->horizontalHeader()->sortIndicatorSection();
    auto ord = table->horizontalHeader()->sortIndicatorOrder();
    if (column != 7) {
        table->sortItems(column, Qt::DescendingOrder);
        table->sortItems(column, Qt::AscendingOrder);
        table->sortItems(column, ord);
    }
}

void MyWidget::LoadClicked() {
    //table->setSortingEnabled(false);
    QFile file("input.txt");
    if (file.open(QIODevice::ReadOnly)) {
        QTextStream inp(&file);
        QString line;
        while (!inp.atEnd()) {
            line = inp.readLine();
            QStringList fields = line.split(",");
            AddPerson(fields);
        }
    }
    file.close();
    int column = table->horizontalHeader()->sortIndicatorSection();
    auto ord = table->horizontalHeader()->sortIndicatorOrder();
    if (column != 7) {
        table->sortItems(column, Qt::DescendingOrder);
        table->sortItems(column, Qt::AscendingOrder);
        table->sortItems(column, ord);
    }
}

void MyWidget::SaveClicked() {
    if (terran.size()) {
        QFile file("output.txt");
        if (file.open(QIODevice::Truncate |
            QIODevice::ReadWrite)) {
            QTextStream out(&file);
            QString line;
            for (int i = 0; i < terran.size(); i++) {
                out << terran[i] << '\n';
            }
        }
    }
}

```

```

        }
    }
    file.close();
}

void MyWidget::DelClicked() {
    QItemSelectionModel* select = table->selectionModel();
    if (select->hasSelection()) {
        int row = select->selectedRows()[0].row();
        terran.removeAt(row);
        table->removeRow(row);
    }
}

void MyWidget::UpdAfterSort(int col, Qt::SortOrder order) {
    std::sort(terran.begin(), terran.end(), SortFunctor(col, order));
}

void MyWidget::LEUUpdate(const QString& text) {
    if (!text.isEmpty()) {
        btnSearch->setEnabled(true);
    }
    else {
        btnSearch->setEnabled(false);
        btnNext->setEnabled(false);
        btnPrev->setEnabled(false);
    }
}

void MyWidget::SearchClicked() {
    found = table->findItems(LiEdSearch->text(), Qt::MatchExactly);
    iterFoun = found.begin();
    if (iterFoun != found.end()) {
        table->selectRow((*iterFoun)->row());
        if ((iterFoun + 1) != found.end()) {
            btnNext->setEnabled(true);
        }
    }
}

void MyWidget::NextClicked() {
    iterFoun++;
    table->selectRow((*iterFoun)->row());
    btnPrev->setEnabled(true);
}

```

```

        if ((iterFoun + 1) != found.end()) {
            btnNext->setEnabled(true);
        }
        else {
            btnNext->setEnabled(false);
        }
    }
}

void MyWidget::PrevClicked() {
    iterFoun--;
    table->selectRow((*iterFoun)->row());
    btnNext->setEnabled(true);
    if ((iterFoun) != found.begin()) {
        btnPrev->setEnabled(true);
    }
    else {
        btnPrev->setEnabled(false);
    }
}

SortFunctor::SortFunctor(int c, Qt::SortOrder o) : col(c), order(o) {

}

bool SortFunctor::operator()(MyTerran& p1, MyTerran& p2) {
    if (col != 3) { // != Birth
        if (order == Qt::AscendingOrder) {
            return p1.GetAll()[col] < p2.GetAll()[col];
        }
        else {
            return p1.GetAll()[col] > p2.GetAll()[col];
        }
    }
    else {
        if (order == Qt::AscendingOrder) {
            return QDate::fromString(p1.GetBirth(),
::DateMask) < QDate::fromString(p2.GetBirth(),
::DateMask);
        }
        else {
            return QDate::fromString(p1.GetBirth(),
::DateMask) > QDate::fromString(p2.GetBirth(),
::DateMask);
        }
    }
}

```

```

    }
}
}

```

## MyTerran.h

```

#include <QTextStream>
#include <QDate>
#include <QString>
#include <QStringList>
#include "GlobVar.h"
class MyTerran {
private:
    QString TerLaN;
    QString TerFiN;
    QString TerSeN;
    QString TerBirth;
    QString TerAddress;
    QString TerEmail;
    QString TerPhone;
public:
    MyTerran(QString lastName = "",
              QString firstName = "",
              QString secName = "",
              QString birth = "02.01.1903",
              QString address = "",
              QString email = "",
              QString phone = ""
    );
    MyTerran(QStringList list);
    QString GetLastName();
    QString GetFirstName();
    QString GetSecName();
    QString GetBirth();
    QString GetAddress();
    QString GetEmail();
    QString GetPhone();
    QStringList GetAll() ;
    bool operator==(const MyTerran& p) const;
    friend QTextStream& operator<<(QTextStream& s, const MyTerran& p);
};

```

```
QTextStream& operator<<(QTextStream& stream, const MyTerran& p);
```

## MyTerran.cpp

```
#include "MyTerran.h"
```

```
MyTerran::MyTerran(QString lastName, QString firstName, QString secName,  
QString birth, QString address, QString email, QString phone)
```

```
    : TerLaN(lastName),  
    TerFiN(firstName),  
    TerSeN(secName),  
    TerBirth(birth),  
    TerAddres(address),  
    TerEmail(email),  
    TerPhone(phone)
```

```
{}
```

```
MyTerran::MyTerran(QStringList list)
```

```
    : TerLaN(list[0]),  
    TerFiN(list[1]),  
    TerSeN(list[2]),  
    TerBirth(list[3]),  
    TerAddres(list[4]),  
    TerEmail(list[5]),  
    TerPhone(list[6])
```

```
{}
```

```
QString MyTerran::GetLastName() {  
    return TerLaN;
```

```
}
```

```
QString MyTerran::GetFirstName() {  
    return TerFiN;
```

```
}
```

```
QString MyTerran::GetSecName() {  
    return TerSeN;
```

```
}
```

```
QString MyTerran::GetBirth() {  
    return TerBirth;
```

```
}
```

```
QString MyTerran::GetAddress() {  
    return TerAddres;
```

```
}
```

```
QString MyTerran::GetEmail() {
```



```

        return TerEmail;
}
QString MyTerran::GetPhone() {
    return TerPhone;
}
QStringList MyTerran::GetAll() {
    return QStringList() << TerLaN << TerFiN << TerSeN << TerBirth <<
}
bool MyTerran::operator==(const MyTerran& p) const {
    return (TerLaN == p.TerLaN) && (TerFiN == p.TerFiN) &&
        (TerSeN == p.TerSeN) && (TerBirth == p.TerBirth) &&
        (TerAddres == p.TerAddres) && (TerEmail == p.TerEmail) &&
            (TerPhone == p.TerPhone);
}
QTextStream& operator<<(QTextStream& stream, const MyTerran& p) {
    stream << p.TerLaN << ", "
        << p.TerFiN << ", "
        << p.TerSeN << ", "
        << p.TerBirth << ", "
        << p.TerAddres << ", "
        << p.TerEmail << ", "
        << p.TerPhone;
    return stream;
}

```

## AddNew.h

```

#include <QGridLayout>
#include <QLabel>
#include <QPushButton>
#include <QString>
#include <QStringList>
#include <QDialog>
#include <QLineEdit>
#include <QRegExp>
#include <QRegExpValidator>
#include <QDateEdit>
#include <QMessageBox>
#include "GlobVar.h"
#include "MyTerran.h"
class AddNew : public QDialog {

```

```

        Q_OBJECT
private:
    QGridLayout* grid;
    QLabel* lblLN;
    QLabel* lblFN;
    QLabel* lblSN;
    QLabel* lblBirth;
    QLabel* lblAddress;
    QLabel* lblEmail;
    QLabel* lblPhone;
    QLabel* temaplatePhone;
    QLineEdit* LiEdLN;
    QLineEdit* LiEdFN;
    QLineEdit* LiEdSN;
    QLineEdit* LiEdAddress;
    QLineEdit* LiEdEmail;
    QLineEdit* LiEdPhone;
    QDateEdit* DaEdBirth;
    QPushButton* btnOk;
    QPushButton* btnCancel;
    QRegExp regexName;
    QRegExp regexEmail;
    QRegExp regexPhone;
    MyTerran inpPerson;
    MyTerran* outPerson;
    QStringList notFilled;
    QStringList notValidated;
    bool IsAllFilled();
    bool ValidateInput();
    void ThrowWarning(QString msg, QStringList& list);
public:
    AddNew(QString title, QWidget* parent = Q_NULLPTR);
    ~AddNew();
    MyTerran* GetOutputPerson();
public slots:
    void accept() override;
};

```

## AddNew.cpp

```
#include "AddNew.h"
```

```
AddNew::AddNew(QString title, QWidget* parent)
```

```
    : QDialog(parent),  
    regexName(::fslname_regex),  
    regexEmail(::email_regex),  
    regexPhone(::phone_regex),  
    inpPerson(MyTerran()),  
    outPerson(nullptr),  
    notFilled(),  
    notValidated()
```

```
{
```

```
    this->setWindowTitle(title);  
    this->resize(600, 600);  
    this->setFixedSize(600, 600);  
    grid = new QGridLayout(this);  
    lblLN = new QLabel("Last name :", this);  
    lblFN = new QLabel("First name :", this);  
    lblSN = new QLabel("Second name :", this);  
    lblBirth = new QLabel("Date of birth :", this);  
    lblAddress = new QLabel("Address :", this);  
    lblEmail = new QLabel("E-mail :", this);  
    lblPhone = new QLabel("Phone :", this);  
    temaplatePhone = new QLabel("+X(XXX)XXX-XX-XX");  
    LiEdLN = new QLineEdit(inpPerson.GetLastName(), this);  
    LiEdFN = new QLineEdit(inpPerson.GetFirstName(), this);  
    LiEdSN = new QLineEdit(inpPerson.GetSecName(), this);  
    DaEdBirth = new QDateEdit(QDate::fromString(inpPerson.GetBirth(),  
        ::DateMask), this);  
    DaEdBirth->setMinimumDate(QDate::fromString("02.01.1903",  
        ::DateMask));  
    DaEdBirth->setMaximumDate(QDate::currentDate());  
    LiEdAddress = new QLineEdit(inpPerson.GetAddress(), this);  
    LiEdEmail = new QLineEdit(inpPerson.GetEmail(), this);  
    LiEdPhone = new QLineEdit(inpPerson.GetPhone(), this);  
    btnOk = new QPushButton("OK", this);  
    btnCancel = new QPushButton("Cancel", this);  
    grid->addWidget(lblLN, 0, 0, 1, 5);  
    grid->addWidget(lblFN, 1, 0, 1, 5);  
    grid->addWidget(lblSN, 2, 0, 1, 5);
```

```

grid->addWidget(lblBirth, 3, 0, 1, 5);
grid->addWidget(lblAddress, 4, 0, 1, 5);
grid->addWidget(lblEmail, 5, 0, 1, 5);
grid->addWidget(lblPhone, 6, 0, 1, 5);
grid->addWidget(temaplatePhone, 7, 1, 1, 5);
grid->addWidget(LiEdLN, 0, 1, 1, 5);
grid->addWidget(LiEdFN, 1, 1, 1, 5);
grid->addWidget(LiEdSN, 2, 1, 1, 5);
grid->addWidget(DaEdBirth, 3, 1, 1, 5);
grid->addWidget(LiEdAddress, 4, 1, 1, 5);
grid->addWidget(LiEdEmail, 5, 1, 1, 5);
grid->addWidget(LiEdPhone, 6, 1, 1, 5);
grid->addWidget(btnOk, 7, 4, 1, 1);
grid->addWidget(btnCancel, 7, 5, 1, 1);
connect(btnOk, SIGNAL(clicked()), SLOT(accept()));
connect(btnCancel, SIGNAL(clicked()), SLOT(reject()));
QRegExpValidator* validatorName = new QRegExpValidator(regexName,
this);
LiEdLN->setValidator(validatorName);
LiEdFN->setValidator(validatorName);
LiEdSN->setValidator(validatorName);
QRegExpValidator* validatorEmail =
new QRegExpValidator(regexEmail, this);
LiEdEmail->setValidator(validatorEmail);
QRegExpValidator* validatorPhone =
new QRegExpValidator(regexPhone, this);
LiEdPhone->setValidator(validatorPhone);
}
AddNew::~~AddNew() {delete outPerson;}
MyTerran* AddNew::GetOutputPerson() {return outPerson; }
bool AddNew::IsAllFilled() {
    QLineEdit* tmp = new QLineEdit(DaEdBirth->text());
    QList<QLineEdit*> widgets = { LiEdLN, LiEdFN, LiEdSN, tmp,
    LiEdAddress, LiEdEmail, LiEdPhone };
    for (int i = 0; i < widgets.size(); i++) {
        if (widgets[i]->text() == "" || widgets[i]->text()
== "02.01.1903") {
            if (i == 0)
                notFilled << "Last name";
            else if (i == 1)

```

```

        notFilled << "First name";
    else if (i == 2)
        notFilled << "Second name";
    else if (i == 3)
        notFilled << "Date of birth";
    else if (i == 4)
        notFilled << "Address";
    else if (i == 5)
        notFilled << "E-mail";
    else if (i == 6)
        notFilled << "Phone";
    }
}
delete tmp;
return (notFilled.size() == 0);
}
bool AddNew::ValidateInput() {
    if (!regexName.exactMatch(LiEdLN->text()))
        notValidated << "Last name";
    if (!regexName.exactMatch(LiEdFN->text()))
        notValidated << "First name";
    if (!regexName.exactMatch(LiEdSN->text()))
        notValidated << "Second name";
    if (!regexEmail.exactMatch(LiEdEmail->text()))
        notValidated << "E-mail";
    if (!regexPhone.exactMatch(LiEdPhone->text()))
        notValidated << "Phone";
    return (notValidated.size() == 0);
}
void AddNew::ThrowWarning(QString msg, QStringList& list) {
    int high = list.size();
    for (int i = 0; i < high - 1; i++) {
        msg += list[i] + ", ";
    }
    msg += list[high - 1];

    QMessageBox* msgBox = new QMessageBox(this);
    msgBox->setWindowTitle("Warning");
    msgBox->setIcon(QMessageBox::Warning);
    msgBox->setText(msg);
}

```

```

        msgBox->exec();
        list.clear();
    }
    void AddNew::accept() {
        if (IsAllFilled()) {
            if (ValidateInput()) {
                outPerson = new MyTerran(LiEdLN->text(),
                    LiEdFN->text(), LiEdSN->text(), DaEdBirth->text(),
                    LiEdEmail->text(), LiEdPhone->text() );
                QDialog::accept();
                this->close();
            }
            else
                ThrowWarning("The strings are filled incorrectly:\n",
                    notValidated);
        }
        else {
            ThrowWarning("Please fill the following:\n", notFilled);
        }
    }
}

```

## EditLine.h

```

#include <QGridLayout>
#include <QLabel>
#include <QPushButton>
#include <QDialog>
#include <QString>
#include <QStringList>
#include <QLineEdit>
#include <QRegExp>
#include <QRegExpValidator>
#include <QDateEdit>
#include <QMessageBox>
#include "GlobVar.h"
#include "MyTerran.h"
class EditLine : public QDialog {
    Q_OBJECT
private:
    QGridLayout* grid;
    QLabel* lblLN;

```

```

    QLabel* lblFN;
    QLabel* lblSN;
    QLabel* lblBirth;
    QLabel* lblAddress;
    QLabel* lblEmail;
    QLabel* lblPhone;
    QLabel* temaplatePhone;
    QLineEdit* LiEdLN;
    QLineEdit* LiEdFN;
    QLineEdit* LiEdSN;
    QLineEdit* LiEdAddress;
    QLineEdit* LiEdEmail;
    QLineEdit* LiEdPhone;
    QDateEdit* DaEdBirth;
    QPushButton* btnOk;
    QPushButton* btnCancel;
    QRegExp regexName;
    QRegExp regexEmail;
    QRegExp regexPhone;
    MyTerran* inpPerson;
    MyTerran* outPerson;
    QStringList notFilled;
    QStringList notValidated;
    bool IsAllFilled();
    bool IsEdited();
    bool ValidateInput();
    void ThrowWarning(QString msg, QStringList& list);
public:
    EditLine(QString title, MyTerran* inpPerson,
              QWidget* parent = Q_NULLPTR);
    ~EditLine();
    MyTerran* GetOutputPerson();
public slots:
    void accept() override;
};

```

## EditLine.cpp

```

#include "EditLine.h"
EditLine::EditLine(QString title, MyTerran* p, QWidget* parent)
    : QDialog(parent),

```

```

regexName(::fslname_regex),
regexEmail(::email_regex),
regexPhone(::phone_regex),
inpPerson(p),
outPerson(nullptr),
notFilled(),
notValidated()
{
    this->setWindowTitle(title);
    this->resize(600, 600);
    this->setFixedSize(600, 600);
    grid = new QGridLayout(this);
    lblLN = new QLabel("Last name :", this);
    lblFN = new QLabel("First name :", this);
    lblSN = new QLabel("Second name :", this);
    lblBirth = new QLabel("Date of birth :", this);
    lblAddress = new QLabel("Address :", this);
    lblEmail = new QLabel("E-mail :", this);
    lblPhone = new QLabel("Phone :", this);
    temaplatePhone = new QLabel("+X(XXX)XXX-XX-XX");
    LiEdLN = new QLineEdit(inpPerson->GetLastName(), this);
    LiEdFN = new QLineEdit(inpPerson->GetFirstName(), this);
    LiEdSN = new QLineEdit(inpPerson->GetSecName(), this);
    DaEdBirth = new QDateEdit(QDate::fromString(inpPerson->GetBirth(),
        ::DateMask), this);
    DaEdBirth->setMinimumDate(QDate::fromString("02.01.1903",
        ::DateMask));
    DaEdBirth->setMaximumDate(QDate::currentDate());
    LiEdAddress = new QLineEdit(inpPerson->GetAddress(), this);
    LiEdEmail = new QLineEdit(inpPerson->GetEmail(), this);
    LiEdPhone = new QLineEdit(inpPerson->GetPhone(), this);
    btnOk = new QPushButton("OK", this);
    btnCancel = new QPushButton("Cancel", this);
    grid->addWidget(lblLN, 0, 0, 1, 5);
    grid->addWidget(lblFN, 1, 0, 1, 5);
    grid->addWidget(lblSN, 2, 0, 1, 5);
    grid->addWidget(lblBirth, 3, 0, 1, 5);
    grid->addWidget(lblAddress, 4, 0, 1, 5);
    grid->addWidget(lblEmail, 5, 0, 1, 5);
    grid->addWidget(lblPhone, 6, 0, 1, 5);

```



```

grid->addWidget(temaplatePhone, 7, 1, 1, 5);
grid->addWidget(LiEdLN, 0, 1, 1, 5);
grid->addWidget(LiEdFN, 1, 1, 1, 5);
grid->addWidget(LiEdSN, 2, 1, 1, 5);
grid->addWidget(DaEdBirth, 3, 1, 1, 5);
grid->addWidget(LiEdAddress, 4, 1, 1, 5);
grid->addWidget(LiEdEmail, 5, 1, 1, 5);
grid->addWidget(LiEdPhone, 6, 1, 1, 5);
grid->addWidget(btnOk, 7, 4, 1, 1);
grid->addWidget(btnCancel, 7, 5, 1, 1);
connect(btnOk, SIGNAL(clicked()), SLOT(accept()));
connect(btnCancel, SIGNAL(clicked()), SLOT(reject()));
QRegExpValidator* validatorName =
    new QRegExpValidator(regexName, this);
LiEdLN->setValidator(validatorName);
LiEdFN->setValidator(validatorName);
LiEdSN->setValidator(validatorName);
QRegExpValidator* validatorEmail =
    new QRegExpValidator(regexEmail, this);
LiEdEmail->setValidator(validatorEmail);
QRegExpValidator* validatorPhone =
    new QRegExpValidator(regexPhone, this);
LiEdPhone->setValidator(validatorPhone);
}
EditLine::~EditLine() {delete outPerson;}
MyTerran* EditLine::GetOutputPerson() {return outPerson;}
bool EditLine::IsAllFilled() {
    QLineEdit* tmp = new QLineEdit(DaEdBirth->text());
    QList<QLineEdit*> widgets = { LiEdLN, LiEdFN, LiEdSN, tmp,
    LiEdAddress, LiEdEmail, LiEdPhone };
    for (int i = 0; i < widgets.size(); i++) {
        if (widgets[i]->text() == "" |
            widgets[i]->text() == "02.01.1903") {
            if (i == 0)
                notFilled << "Last name";
            else if (i == 1)
                notFilled << "First name";
            else if (i == 2)
                notFilled << "Second name";
            else if (i == 3)

```

```

        notFilled << "Date of birth";
    else if (i == 4)
        notFilled << "Address";
    else if (i == 5)
        notFilled << "E-mail";
    else if (i == 6)
        notFilled << "Phone";
    }
}
delete tmp;
return (notFilled.size() == 0);
}
bool EditLine::IsEdited() {return !(*inpPerson == *outPerson);}
bool EditLine::ValidateInput() {
    if (!regexName.exactMatch(LiEdLN->text()))
        notValidated << "Last name";
    if (!regexName.exactMatch(LiEdFN->text()))
        notValidated << "First name";
    if (!regexName.exactMatch(LiEdSN->text()))
        notValidated << "Second name";
    if (!regexEmail.exactMatch(LiEdEmail->text()))
        notValidated << "E-mail";
    if (!regexPhone.exactMatch(LiEdPhone->text()))
        notValidated << "Phone";
    return (notValidated.size() == 0);
}
void EditLine::ThrowWarning(QString msg, QStringList& list) {
    int high = list.size();
    for (int i = 0; i < high - 1; i++) {
        msg += list[i] + ", ";
    }
    msg += list[high - 1];
    QMessageBox* msgBox = new QMessageBox(this);
    msgBox->setWindowTitle("Warning");
    msgBox->setIcon(QMessageBox::Warning);
    msgBox->setText(msg);
    msgBox->exec();
    list.clear();
}
void EditLine::accept() {

```

```

if (IsAllFilled()) {
    if (ValidateInput()) {
        outPerson = new MyTerran(LiEdLN->text(),
            LiEdFN->text(), LiEdSN->text(),
            DaEdBirth->text(), LiEdAddress->text(),
            LiEdEmail->text(), LiEdPhone->text());
        if (IsEdited())
            QDialog::accept();
        else
            QDialog::reject();
        this->close();
    }
    else
        ThrowWarning("The strings are filled incorrectly:\n",
            notValidated);
}
else
    ThrowWarning("Please fill the following:\n", notFilled);
}

```

## GlobVar.h

```

#define ColumnCount 7
namespace {
    static const char* DateMask = "dd.MM.yyyy";

    static const char* fslname_regex =
        "(^[A-Z])([A-Z|a-z|0-9| |-]*)([A-Z|a-z|0-9]$)";
    static const char* email_regex =
        "^[a-z|A-Z|0-9|+|_|@|.|-]+@[a-z|A-Z|0-9|+|_|@|.|-]+[a-z|A-Z|0-9|+|_|@|.|-]+$";
    static const char* phone_regex =
        "\\+\\d{1,3}\\(\\d{3}\\)\\d{3}\\-\\d{2}\\-\\d{2}";
}

```