Санкт-Петербургский Политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа искусственного интеллекта

Математическое обеспечение и администрирование информационных систем

**Отчет по курсовой работе по объектно-ориентированному программированию**

**«Телефонный справочник»**

Выполнила: Гаврилова Вероника 5130203/30002

Санкт-Петербург, 2024

**Введение**

Цель: написать телефонный справочник на С++ с использованием библиотеки Qt и базы данных.

Задачи:

1. Написать программу без Qt, данные хранить в файле;
2. Добавить графический интерфейс с использованием Qt. Для хранения использовать QFile.
3. Не удаляя хранение в файле добавить возможность хранения в базе данных.

Так же есть обязательные условия осуществления данной программы.

1. Обязательные поля для хранения: Имя, Фамилия, Отчество, адрес, дата рождения, email, телефонные номера (рабочий, домашний, служебный) в любом количестве.Организовать проверку всех вводимых данных на корректность при помощи регулярных выраженийФамилия, Имя или Отчество --- должны содержать только буквы и цифры различных алфавитов, а также дефис и пробел, но при этом должны начинаться только на буквы, и не могли бы оканчиваться или начинаться на дефис. Все незначимые пробелы перед и после данных должны удаляться.

Варианты записи телефона:

+78121234567

88121234567

+7(812)1234567

8(812)1234567

+7(812)123-45-67

8(812)123-45-67

1. Дата рождения должна быть меньше текущей даты, число месяцев в дате должно быть от 1 до 12, число дней от 1 до 31, причем должно учитываться различное число дней в месяце и високосные года
2. E-mail должен содержать в себе имя пользователя состоящее из латинских букв и цифр, символ разделения пользователя и имени домена(@), а также сам домен состоящий из латинских букв и цифр. Все незначащие пробелы (включая пробелы перед и после символа @) должны быть удалены.
3. Приложение должно позволять добавлять/удалять записи, а также редактировать все поля. Осуществлять сортировку отображаемых данных по указанному полю и поиск по нескольким полям.

**1 Этап: запись в файл.**

На этом этапе была осуществлена программа с записью информации в файл. Приведу некоторые фрагменты и функции кода.

Функции для проверки корректности полей ввода:

bool ContactManager::isValidName(const wstring& name)

{

wregex nameRegex(L"^[A-Za-zА-Яа-яЁё][A-Za-zА-Яа-яЁё0-9\\s-]\*[A-Za-zА-Яа-яЁё0-9]$");

return regex\_match(name, nameRegex);

}

bool ContactManager::isValidName(const string& name)

{

regex nameRegex("^[A-Za-zА-Яа-яЁё][A-Za-zА-Яа-яЁё0-9\\s-]\*[A-Za-zА-Яа-яЁё0-9]$");

return regex\_match(name, nameRegex);

}

bool ContactManager::isValidEmail(const string& email)

{

regex emailRegex(R"((\w+)(\.{0,1}\w+)\*@(\w+)(\.\w+)+)");

return regex\_match(email, emailRegex);

}

bool ContactManager::isValidPhone(const string& phone)

{

regex phoneRegex(R"(\+?\d{1,3}[- ]?\(?\d{1,4}?\)?[- ]?\d{1,4}[- ]?\d{1,4})");

return regex\_match(phone, phoneRegex);

}

bool ContactManager::isValidDate(const string& date)

{

tm tm = {};

istringstream ss(date);

ss >> get\_time(&tm, "%Y-%m-%d");

if (ss.fail()) return false;

time\_t t = time(nullptr);

std::tm now;

localtime\_s(&now, &t);

if (tm.tm\_year > now.tm\_year ||

(tm.tm\_year == now.tm\_year && tm.tm\_mon > now.tm\_mon) ||

(tm.tm\_year == now.tm\_year && tm.tm\_mon == now.tm\_mon && tm.tm\_mday > now.tm\_mday)) {

return false;

}

return true;

}

Для проверки текстовых полей были использованы регулярные выражения.

Функция для сохранения контакта в файл:

void ContactManager::saveContacts()

{

ofstream file(filename);

for (const auto& contract : contacts) {

file << contract.firstName << ","

<< contract.lastName << ","

<< contract.middleName << ","

<< contract.address << ","

<< contract.birthDate << ","

<< contract.email << ",";

for (const auto& phone : contract.phoneNumbers) {

file << phone << " ";

}

file << endl;

}

file.close();

}

В начальном меню пользователь выбирает вариант, который он хочет осуществить. Этот вариант осуществлен с помощью switch case.

int main() {

setlocale(LC\_ALL, "Russian");

ContactManager manager;

manager.loadContacts();

while (true) {

cout << "1. Добавить контакт\n"

<< "2. Показать контакты\n"

<< "3. Удалить контакт\n"

<< "4. Редактировать контакт\n"

<< "5. Поиск контактов\n"

<< "6. Сортировка\n"

<< "7. Выход\n"

<< "Выберите действие: ";

int choice;

cin >> choice;

cin.ignore();

switch (choice) {

case 1:

manager.addContact();

break;

case 2:

manager.displayContacts();

break;

case 3:

manager.deleteContact();

break;

case 4:

manager.editContact();

break;

case 5:

manager.searchContacts();

break;

case 6:

manager.sortContacts();

break;

case 7:

return 0;

default:

cout << "Некорректный выбор!" << endl;

}

}

В первом этапе реализованы все необходимые функции:

* добавление контакта
* вывод информации о контакте на экран
* удаление контакта
* редактирование всех полей контакта
* поиск контактов
* сортировка по выбранному полю

**2 Этап: осуществление графического интерфейса**

Для этого создаем файлы addressbook.cpp и addressbook.h для осуществления графического интерфейса.

В классе AddressBook созданы текстовые поля для ввода информации о контакте:

QLineEdit\* firstnameLine;

QLineEdit\* lastnameLine;

QLineEdit\* middlenameLine;

QLineEdit\* addressText;

QLineEdit\* birthdate;

QLineEdit\* email;

QTextEdit\* numbers;

Так же определены кнопки для осуществления заданных функций удаления, добавления, сортировки и остальных.

QPushButton\* addButton;

QPushButton\* displayButton;

QPushButton\* deleteButton;

QPushButton\* editButton;

QPushButton\* searchButton;

QPushButton\* sortButton;

Так же написана функция для отчистки лишних пробелов в конце и начале строки:

string AddressBook::trim(const string& str) {

size\_t start = str.find\_first\_not\_of(" \t\n\r");

size\_t end = str.find\_last\_not\_of(" \t\n\r");

return (start == string::npos) ? "" : str.substr(start, end - start + 1);

}

mainLayout->addWidget(firstnameLabel, 0, 0);

mainLayout->addWidget(firstnameLine, 0, 1);

mainLayout->addWidget(lastnameLabel, 1, 0);

mainLayout->addWidget(lastnameLine, 1, 1);

mainLayout->addWidget(middlenameLabel, 2, 0);

mainLayout->addWidget(middlenameLine, 2, 1);

mainLayout->addWidget(addressLabel, 3, 0, Qt::AlignTop);

mainLayout->addWidget(addressText, 3, 1);

mainLayout->addWidget(birthdateLabel, 4, 0);

mainLayout->addWidget(birthdate, 4, 1);

mainLayout->addWidget(emailLabel, 5, 0);

mainLayout->addWidget(email, 5, 1);

mainLayout->addWidget(numbersLabel, 6, 0);

mainLayout->addWidget(numbers, 6, 1);

mainLayout->addWidget(addButton, 7, 0);

mainLayout->addWidget(displayButton, 7, 1);

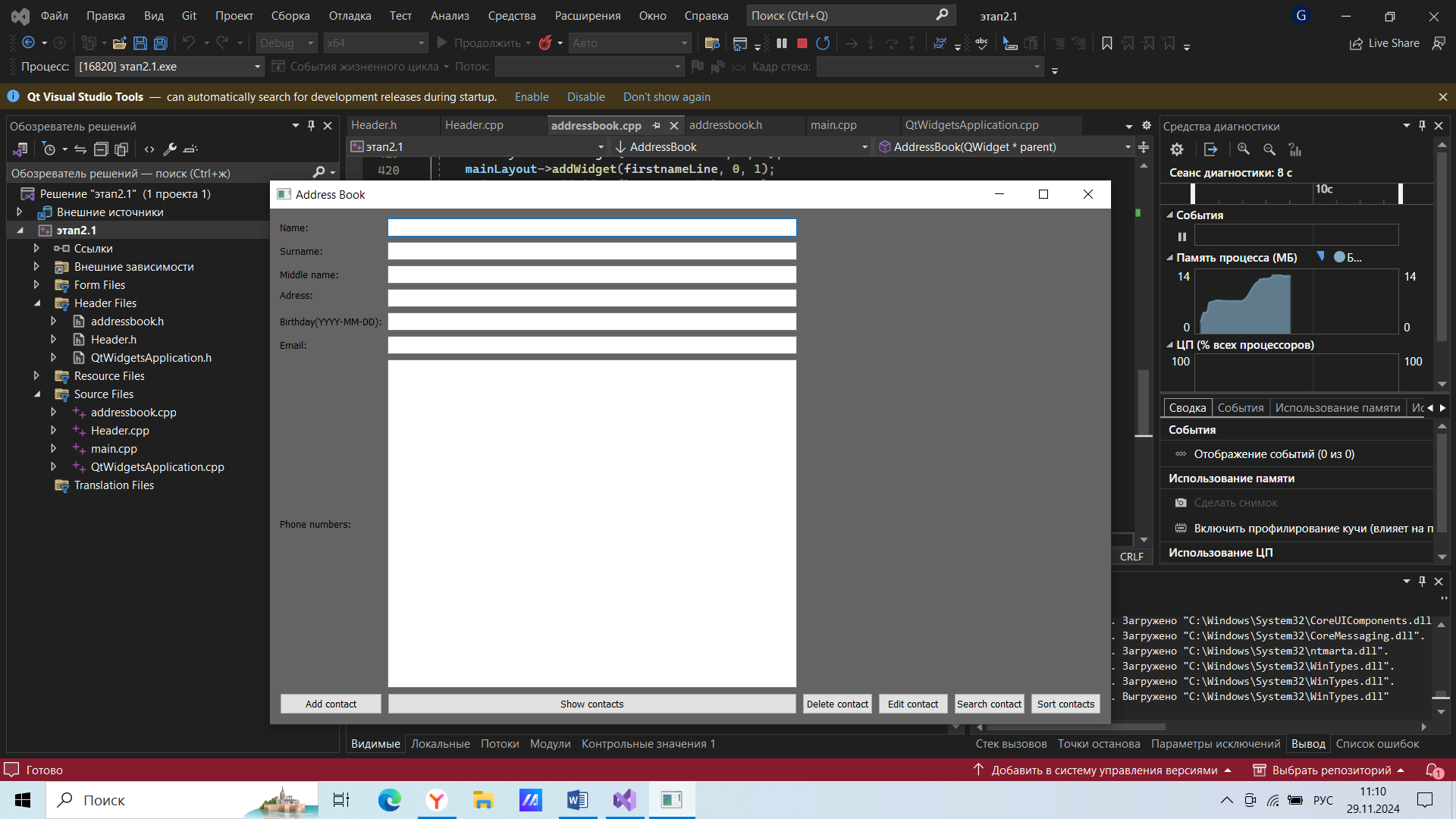
mainLayout->addWidget(deleteButton, 7, 2);

mainLayout->addWidget(editButton, 7, 3);

mainLayout->addWidget(searchButton, 7, 4);

mainLayout->addWidget(sortButton, 7, 5);

В этом фрагменте кода распределяем кнопки и поля для ввода по экрану.



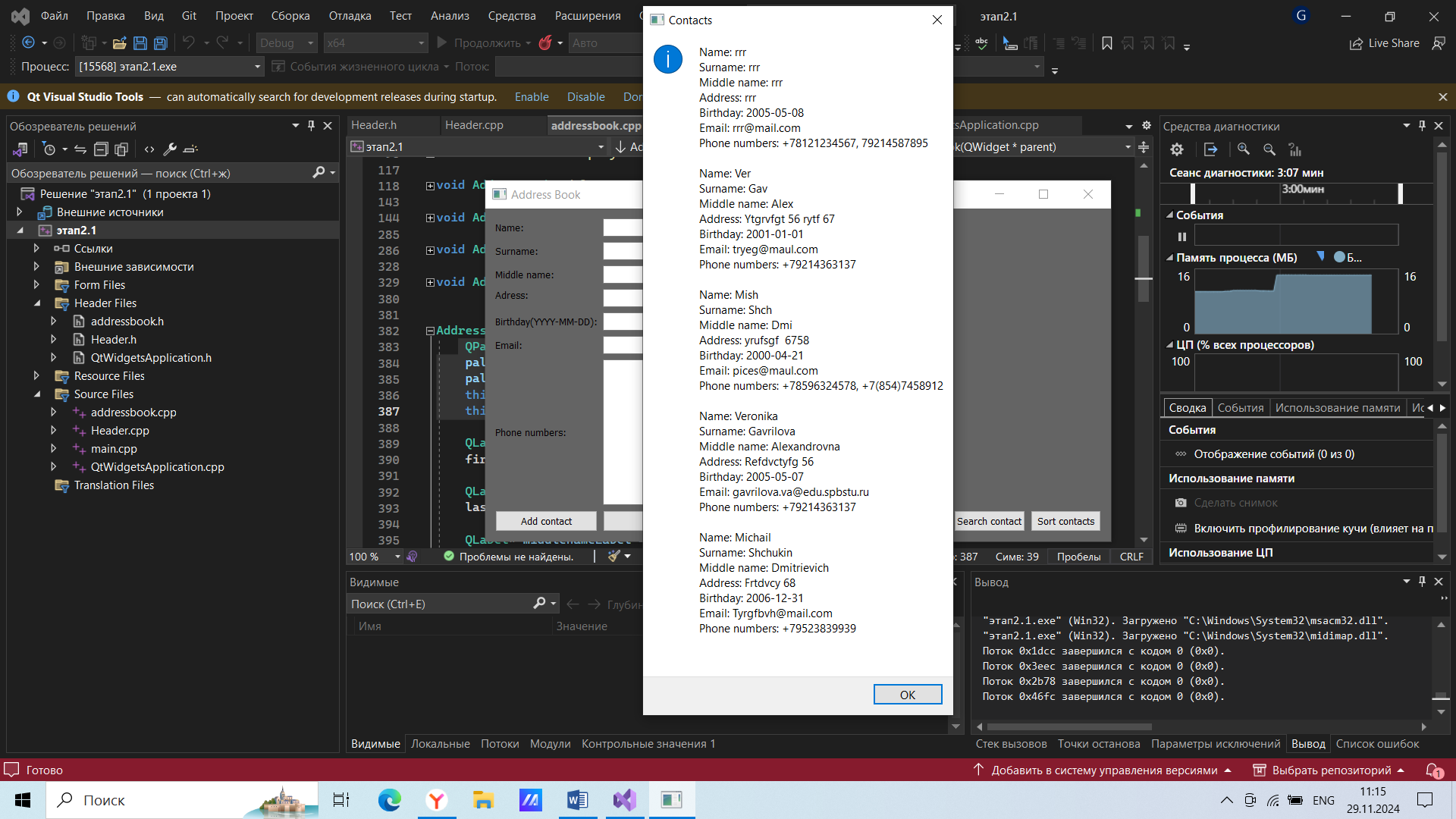
QPalette palette = this->palette();

palette.setColor(QPalette::Window, QColor(100, 100, 100)); palette.setColor(QPalette::WindowText, Qt::black);

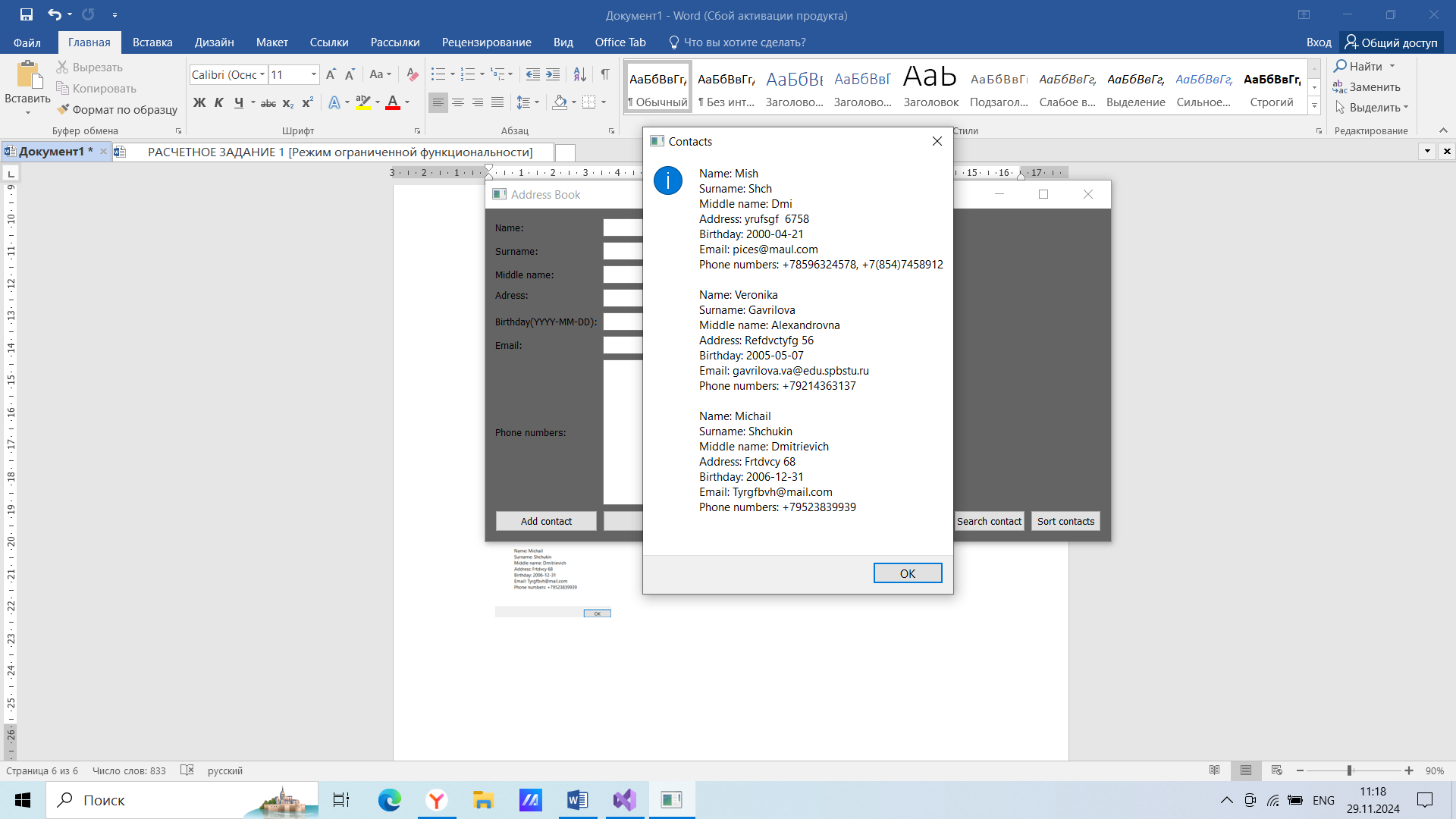
this->setPalette(palette);

this->setAutoFillBackground(true);

Этим фрагментом кода можем устанавливать и менять цвет фона и текста окна.

Контакты выводятся в таком виде: Вывод контактов после удаления некоторых и

и сортировке по возрасту:



Так же было осуществлено хранение и чтение информации QFile.

void ContactManager::saveToFile(const QString& filename) const

{

QFile file(filename);

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

throw std::runtime\_error("Unable to open file for writing");

}

QTextStream out(&file);

for (const auto& contact : contacts) {

QStringList phoneNumbersList;

for (const auto& phoneNumber : contact.phoneNumbers) {

phoneNumbersList.append(QString::fromStdString(phoneNumber));

}

QString phoneNumbers = phoneNumbersList.join(',');

out << QString::fromStdString(contact.firstName) << ";"

<< QString::fromStdString(contact.lastName) << ";"

<< QString::fromStdString(contact.middleName) << ";"

<< QString::fromStdString(contact.address) << ";"

<< QString::fromStdString(contact.birthDate) << ";"

<< QString::fromStdString(contact.email) << ";"

<< phoneNumbers << "\n";

}

file.close();

}

**Этап 3: подключение базы данных.**

После изучения множества баз данных была выбрана SQLite. Её преимущества заключаются в:

* Простой интерфейс в виде командной строки
* Простота установки
* Простота подключения к уже существующей программе

Установить и подключить эту базу данных легко и просто по видео урокам из интернета.

В этом фрагменте кода создается база данных, если она не существует, если же существует, то просто открывается.

void ContactManager::initializeDatabase(const QString& dbFilename) {

sqlite3\* db;

int rc = sqlite3\_open(dbFilename.toStdString().c\_str(), &db);

if (rc) {

throw std::runtime\_error("Cannot open database");

}

std::string sql = R"(

CREATE TABLE IF NOT EXISTS Contacts (

id INTEGER PRIMARY KEY AUTOINCREMENT,

firstName TEXT,

lastName TEXT,

middleName TEXT,

address TEXT,

birthDate TEXT,

email TEXT,

phoneNumbers TEXT

);

)";

char\* errMsg = nullptr;

rc = sqlite3\_exec(db, sql.c\_str(), nullptr, nullptr, &errMsg);

if (rc != SQLITE\_OK) {

std::string error = "SQL error: " + std::string(errMsg);

sqlite3\_free(errMsg);

sqlite3\_close(db);

throw std::runtime\_error(error);

}

sqlite3\_close(db);

}

Следующим этапом следует переписать некоторые функции. Так как не было конкретного задания как осуществить запись в файл и базу данных, то я решила выбрать способ, когда сам программист меняет в коде функции.

Ниже приведу функции, которые были переписаны для корректной работы программы с базой данных.

ContactManager::ContactManager(const QString& dbFilename) {

try {

loadFromDatabase(dbFilename);

}

catch (const exception& e) {

cerr << "Error loading contacts: " << e.what() << endl;

}

}

void ContactManager::saveToDatabase(const QString& dbFilename) const {

sqlite3\* db;

int rc = sqlite3\_open(dbFilename.toStdString().c\_str(), &db);

if (rc) {

throw std::runtime\_error("Cannot open database");

}

std::string sql = R"(

INSERT INTO Contacts (firstName, lastName, middleName, address, birthDate, email, phoneNumbers)

VALUES (?, ?, ?, ?, ?, ?, ?);

)";

sqlite3\_stmt\* stmt;

for (const auto& contact : contacts) {

rc = sqlite3\_prepare\_v2(db, sql.c\_str(), -1, &stmt, nullptr);

if (rc != SQLITE\_OK) {

sqlite3\_close(db);

throw std::runtime\_error("Failed to prepare statement");

}

sqlite3\_bind\_text(stmt, 1, contact.firstName.c\_str(), -1, SQLITE\_TRANSIENT);

sqlite3\_bind\_text(stmt, 2, contact.lastName.c\_str(), -1, SQLITE\_TRANSIENT);

sqlite3\_bind\_text(stmt, 3, contact.middleName.c\_str(), -1, SQLITE\_TRANSIENT);

sqlite3\_bind\_text(stmt, 4, contact.address.c\_str(), -1, SQLITE\_TRANSIENT);

sqlite3\_bind\_text(stmt, 5, contact.birthDate.c\_str(), -1, SQLITE\_TRANSIENT);

sqlite3\_bind\_text(stmt, 6, contact.email.c\_str(), -1, SQLITE\_TRANSIENT);

std::string phoneNumbers = "";

for (const auto& phone : contact.phoneNumbers) {

phoneNumbers += phone + ",";

}

sqlite3\_bind\_text(stmt, 7, phoneNumbers.c\_str(), -1, SQLITE\_TRANSIENT);

rc = sqlite3\_step(stmt);

if (rc != SQLITE\_DONE) {

sqlite3\_finalize(stmt);

sqlite3\_close(db);

throw std::runtime\_error("Failed to execute statement");

}

sqlite3\_finalize(stmt);

}

sqlite3\_close(db);

}

**Вывод**

Выполняя данную курсовую работу были получены знания по работе с фреймворком Qt и базами данных. Были успешно выполнены все поставленные задачи. Программа успешно функционирует и выполняет предписанные ей команды.