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

АУДИОПЛЕЕР С ИСПОЛЬЗОВАНИЕМ ФРЕЙМВОРКА QTБГУИР КП 1-40 02 01 420 ПЗ

|  |  |  |
| --- | --- | --- |
| Студент |  | Группы 150504  А.С. Шалаш |
| Руководитель |  | А.В. Марзалюк |

Минск 2022

Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ЭВМ

*––––––––––––––––––––––––*

(подпись)

–––––––––––––––––––2022 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту    *Шалаш Анне Сергеевне–––––––––––––––––––*

1. Тема проекта   *Аудиоплеер с использованием фреймворка Qt*–––––––– ––––

2. Срок сдачи студентом законченного проекта –––––*15 декабря 2022 г.– —––*

3. Содержание расчетно-пояснительной записки (список вопросов, которые подлежат развитию)

*Введение. 1. Обзор литературы. 2. Структурное проектирование. 3. Функциональное проектирование. 4. Инструкция пользователя. Заключение. Приложение А.  
Приложение Б. Приложение В. Приложение Г. Приложение Д.*

5. Список графического материяла:

*1. Скриншоты работы программы ––––––––––––––––––––––– –––––*

*2. UML-диограмма классов. –––––––––––––––––––––––– ––––*

*3. Схема основных алгоритмов программы. –––––––––––––––––––––– ––––––*

*3. Ведомость документов. . –––––––––––––––––––––––––––––*

6. Консультант по проекту   *А. В. Марзалюк––––––––––––––––––––––––– ––– –*

7. Дата выдачи задания –––––*16 сентября 2022 г. –––––––––––––––––– ––*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

*раздзел 1 до 15 10 – 20 %;–––––––––––––––––––––––––––––––– ––––––*

*разделы 2, 3 до 15 11 – 40 %;–––––––––––––––––––––––––––––––––––––*

*раздел 4 до 30 11 – 20 %;––––––––––––––––––––––––––––––––––––––– –*

*оформление пояснительной записки и графического материала до 15 12 – 20 %*

*Защита курсового проекта с 19 по 30 декабря–––––––––––––––––––––––––––*

РУКОВОДИТЕЛЬ*– ––––––––––– А. В. Марзалюк*

(подпись)

Задание принял к исполнению*–––––––\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_–– А. С. Шалаш*

(дата и подпись студента)

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 7](#_Toc122164107)

[1 ОБЗОР ЛИТЕРАТУРЫ 9](#_Toc122164108)

[1.1 Реализация с использованием библиотеки SFML 9](#_Toc122164109)

[1.2 Реализация с использованием библиотеки BASS 10](#_Toc122164110)

[1.3 Реализация с использованием модуля QtMultimedia 10](#_Toc122164111)

[1.4 Обоснование выбранных методов и алгоритмов 11](#_Toc122164112)

[2 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ 12](#_Toc122164113)

[2.1 Main.cpp 12](#_Toc122164114)

[2.2 Widget.cpp 12](#_Toc122164115)

[2.3 Widget.h 12](#_Toc122164116)

[2.4 Shablon.h 13](#_Toc122164117)

[2.5 Exeption.h 13](#_Toc122164118)

[2.6 Fileexeption.h 13](#_Toc122164119)

[3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 14](#_Toc122164120)

[3.1 Слот void Widget::on\_btn\_add\_clicked() 14](#_Toc122164121)

[3.2 Слот void Widget::on\_Save\_clicked() 15](#_Toc122164122)

[3.3 Слот void Widget::on\_Music\_clicked() 17](#_Toc122164123)

[3.4 Слот void Widget::on\_Lastplaylist\_clicked() 18](#_Toc122164124)

[4 ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ 22](#_Toc122164125)

[ЗАКЛЮЧЕНИЕ 23](#_Toc122164126)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 24](#_Toc122164127)

[ПРИЛОЖЕНИЕ А (обязательное) Листинг кода программного средства 25](#_Toc122164128)

[ПРИЛОЖЕНИЕ Б (обязательное) Скриншоты работы программы 37](#_Toc122164129)

[ПРИЛОЖЕНИЕ В (обязательное) UML-диаграмма классов программного средства 39](#_Toc122164130)

[ПРИЛОЖЕНИЕ Г (обязательное) Блок-схемы алгоритмов функций on\_Clear\_clicked(), DeleteAll(), push() 41](#_Toc122164131)

[ПРИЛОЖЕНИЕ Д (обязательное) Ведомость документов 45](#_Toc122164132)

# ВВЕДЕНИЕ

На сегодняшний день мультимедийные технологии прочно укрепились во многих сферах деятельности. Множество программистов, сценаристов, дизайнеров работают над созданием всё новых и новых проектов.

В рамках проекта необходимо разработать аудиоплеер с использованием фреймворка Qt. Для реализации подобного программного обеспечения был применен язык С++.

Кроссплатформенный инструментарий разработки Qt появился впервые в 1995 году благодаря своим разработчикам Хаарварду Норду и Айрику Чеймб-ингу. С самого начала Qt создавался как программный каркас для создания кроссплатформенных программ с графическим интерфейсом или фреймворк (программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта).

В свою очередь С++ является одним из самых популярных языков программирования и используется для создания различных программ, драйверов, приложений, операционных систем, игр.

Для создания простого аудиоплеера необходимы работа с аудиоустройствами и файлами, а также графическое оформление, что в полной мере обеспечивает фреймворк Qt и язык С++.

Цель курсового проектирования заключается в использовании полученных знаний, умений и навыков в ходе изучения дисциплины «Программирование на языках высокого уровня» и языка программирования с последующей разработкой программного средства для воспроизведения [файлов](https://ru.m.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB) [мультимедиа](https://ru.m.wikipedia.org/wiki/%D0%9C%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0)-содержимого.

Для достижения цели были поставлены следующие задачи: изучить необходимую информацию по алгоритму и разработать программный продукт.

Курсовое проектирование представляет собой учебно-исследовательскую деятельность и необходимо определить объект и предмет исследования. Объектом исследования является воспроизведение аудиофайлов с возможностью остановки и перемотки треков. Предметом исследования является программа «аудиоплеер».

# 1 ОБЗОР ЛИТЕРАТУРЫ

Медиаплеер — компьютерная программа, предназначенная для воспроизведения файлов мультимедиа-содержимого.

Аудиоплеер является одной из разновидностей медиаплееров, предназначенный для воспроизведения только аудиофайлов, как и видеоплеер(воспроизведение только видеофайлов).

Большинство современных операционных систем по умолчанию содержат в своём составе медиаплееры: например, Windows — Windows Media Player, Mac OS X — QuickTime Player (для воспроизведения видео в формате QuickTime) и iTunes (для некоторых других форматов), Linux — Amarok, Rhythmbox или иные (в зависимости от дистрибутива).

Аудиофайлы могут иметь следующие форматы:

* аудиоформаты без сжатия, такие как WAV, AIFF
* аудиоформаты со [сжатием без потерь](https://ru.wikipedia.org/wiki/%D0%A1%D0%B6%D0%B0%D1%82%D0%B8%D0%B5_%D0%B1%D0%B5%D0%B7_%D0%BF%D0%BE%D1%82%D0%B5%D1%80%D1%8C) (APE, FLAC)
* аудиоформаты со [сжатием с потерями](https://ru.wikipedia.org/wiki/%D0%A1%D0%B6%D0%B0%D1%82%D0%B8%D0%B5_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85_%D1%81_%D0%BF%D0%BE%D1%82%D0%B5%D1%80%D1%8F%D0%BC%D0%B8) (MP3, Ogg)

## Реализация с использованием библиотеки SFML

SFML (англ. Simple and Fast Multimedia Library — простая и быстрая мультимедийная библиотека) — свободная кроссплатформенная мультимедийная библиотека. Написана на C++, но доступна также для C, C#, .Net, D, Java, Python, Ruby, OCaml, Go и Rust. Представляет собой объектно-ориентированный аналог SDL.

В настоящее время доступны следующие модули:

* System — управление временем и потоками, он является обязательным, так как все модули зависят от него.
* Window — управление окнами и взаимодействием с пользователем.
* Graphics — делает простым отображение графических примитивов и изображений.
* Audio — предоставляет интерфейс для управления звуком.
* Network — для сетевых приложений.

Библиотека SFML не поддерживает mp3. Только WAV, FLAC, OGG/Vorbis и требует скачивания и установки с сайта разработчика.

## 1.2 Реализация с использованием библиотеки BASS

**BASS (BASS audio library)** — звуковая [библиотека](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), предназначена для использования в [операционных системах](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) [Windows](https://ru.wikipedia.org/wiki/Microsoft_Windows) и [macOS](https://ru.wikipedia.org/wiki/MacOS); также доступна для [Linux](https://ru.wikipedia.org/wiki/Linux), [Win64](https://ru.wikipedia.org/wiki/Windows_API), [WinCE](https://ru.wikipedia.org/wiki/Windows_CE), [Android](https://ru.wikipedia.org/wiki/Android) и [iOS](https://ru.wikipedia.org/wiki/IOS)-платформ.

Цель библиотеки заключается в том, чтобы предоставить [разработчикам](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%87%D0%B8%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) образец аудио-центра, в частности, обеспечить работу с потоками ([MP3](https://ru.wikipedia.org/wiki/MP3), [MP2](https://ru.wikipedia.org/wiki/MP2), [MP1](https://ru.wikipedia.org/wiki/MP1), [OGG](https://ru.wikipedia.org/wiki/OGG), [WAV](https://ru.wikipedia.org/wiki/WAV), [AIFF](https://ru.wikipedia.org/wiki/AIFF)), [MOD](https://ru.wikipedia.org/wiki/MOD) файлами ([XM](https://ru.wikipedia.org/wiki/XM), [IT](https://ru.wikipedia.org/wiki/IT), [S3M](https://ru.wikipedia.org/wiki/S3M), [MOD](https://ru.wikipedia.org/wiki/MOD), [MTM](https://ru.wikipedia.org/w/index.php?title=MTM&action=edit&redlink=1), [UMX](https://ru.wikipedia.org/w/index.php?title=UMX&action=edit&redlink=1)), [MO3](https://ru.wikipedia.org/wiki/MO3) файлами ([mp3](https://ru.wikipedia.org/wiki/Mp3), [ogg](https://ru.wikipedia.org/wiki/Ogg) и сжатых [MOD](https://ru.wikipedia.org/wiki/MOD) файлов), функциями записи, редактирования тегов аудиофайлов, конвертирования популярных форматов, «[граббить](https://ru.wikipedia.org/wiki/CD_ripper)» [AudioCD](https://ru.wikipedia.org/wiki/CDA), потребляя малое количество системных ресурсов.

Весь набор возможностей предоставляется в виде [динамической библиотеки](https://ru.wikipedia.org/wiki/DLL) размером около 100 [Kb](https://ru.wikipedia.org/wiki/%D0%9A%D0%B8%D0%BB%D0%BE%D0%B1%D0%B0%D0%B9%D1%82).

Для работы BASS в операционной системе [Windows](https://ru.wikipedia.org/wiki/Microsoft_Windows), требуется [DirectX](https://ru.wikipedia.org/wiki/DirectX) 3 или выше, библиотека использует [DirectSound](https://ru.wikipedia.org/wiki/DirectSound) и [DirectSound3D](https://ru.wikipedia.org/wiki/DirectSound3D) для воспроизведения и записи звука. В [macOS](https://ru.wikipedia.org/wiki/MacOS) BASS использует [CoreAudio](https://ru.wikipedia.org/w/index.php?title=CoreAudio&action=edit&redlink=1).

## 1.3 Реализация с использованием модуля QtMultimedia

Библиотека Qt — это множество классов (более 500). Qt не является единым целым, она разбита на модули. Иерархия классов Qt имеет четкую внутреннюю структуру. Любая Qt-программа с графическим интерфейсом так или иначе должна использовать хотя бы один трёх модулей: QtCore, QtGui или QtWidgets*.* Для каждого модуля Qt предоставляет отдельный заголовочный файл, содержащий заголовочные файлы всех классов этого модуля.

Помимо одного из графических модулей, для решения поставленной задачи необходимо применение модуля QtMultimedia, который представляет собой средства работы с мультимедийными устройствами и файлами.

## 1.4 Обоснование выбранных методов и алгоритмов

Для выполнения поставленной задачи был выбран метод реализации с помощью модуля QtMultimedia. Этот модуль предоставляет возможность прослушивания аудиофайлов самого распространенного формата mp3(чего достаточно в рамках этого проекта по созданию аудиоплеера) и не требует скачивания и установки дополнительных библиотек из сторонних источников.

# 2 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ

Код программы разбит на модули для более удобного использования и изменения функционала программы. Подробное описание классов см. в UML-диаграмме в Приложении В.

## 2.1 Main.cpp

В этом файле написана main-функция, в которой создается объект класса Widget и вызывается метод его отображения на экране.

## 2.2 Widget.cpp

В файле Widget.cpp описаны конструктор, деструктор и методы класса Widget.

В конструкторе инициализируются объекты для проигрывания аудиофайлов и отображения плейлиста, подключаются кнопки управления к слотам управления, слот круговой шкалы регулировки звука, слоты слайдера трека. Подключается слот установки в плейлист трека при даблклике по нему в таблице, подключается слот установки в специальном лейбле названия файла при изменении индекса текущего трека в плейлисте.

Прописаны функции слотов слайдера, добавления треков, сохранения и удалания плейлиста.

## 2.3 Widget.h

Объявляются конструктор, деструктор, приватные поля ui, m\_playListModel, m\_player, m\_playlist и публичные методы msecsToString(), setting(), on\_btn\_add\_clicked(), on\_Save\_clicked(), on\_Clear\_clicked(), on\_Lastplaylist\_clicked(), on\_Music\_clicked(), slotSetSliderPosition(), slotSetMediaPosition (), slotSetDuration () класса Widget.

## 2.4 Shablon.h

В файле находятся классы List: объявлены приватные поля \*head, \*tail, size и описаны публичные методы DeleteAll(), push(), deleteElement(), getsize(), begin(), end(), класс MyIterator: объявлены приватные поля \*mPointer, index и описаны публичные методы MyIterator (Node<T>\* pointer), setPointer(Node<T> \*newPointer), getPointer() и методы перегруженных операторов, структура Node: объявлены переменные info, \*next и \*prev.

## 2.5 Exeption.h

В файле находятся класс-перечисление Errors с приватными полями кодов ошибок и класс Exception с защищенными полями code, message и виртуальная функция show().

## 2.6 Fileexeption.h

В файле находится класс FileException, который унаследован от класса Exception, содержит приватное поле \*window, публичные конструктор и переопределение метода show().

# 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Для лучшего понимания кода и работы программы ниже приведен пошаговый алгоритм работы функций on\_btn\_add\_clicked(), on\_Save\_clicked(), on\_Music\_clicked(), on\_Lastplaylist\_clicked(). Блок-схемы функций on\_Clear\_clicked(), push(), DeleteAll() приведены в приложении Г(Г1, Г2, Г3 соответственно).

## 3.1 Обработчик void Widget::on\_btn\_add\_clicked()

1. Инициализировать список строк files типа QStringList и присвоить ему значение списка файлов выбранных через вызов диалогового окна QFileDialod;
2. Начать цикл int i = 0, i < files.size(), i++;
3. Вызовом метода push(), передав ему возвращенное по методу at(i) значение элемента списка files под индексом i, произвести добавление элемента в конец списка list;
4. Если i < files.size(), перейти к шагу 3, иначе к шагу 5;
5. Начать блок макроса foreach(){} переработать элементы списка строк files в единую строку filePath типа QString;
6. Объявить список items типа QList с объектами типа QStandartItem\*;
7. Вызовом метода append() поместить в список items значения типа new QStandartItem(QDir(filePath).dirName());
8. Вызовом метода append() поместить в список items значения типа new QStandartItem(filePath);
9. Вызовом метода appendRow(), передав ему список items, установить список в модель данных плейлиста для отображения m\_playListModel;
10. Вызовом метода addMedia(), передав ему значения QUrl(filePath), установить URL-адреса плейлист проигрывателя m\_playlist;
11. Завершить макрос foreach(){};
12. Конец.

## 3.2 Обработчик void Widget::on\_Save\_clicked()

1. Объявить диалог отображения сообщения msg типа QMessageBox;
2. Вызвать метод setText() для msg и передать ему текст основного сообщения "Вы действительно хотите Сохранить Нынешний Плейлист?";
3. Вызвать метод setInformativeText() для msg и передать ему текст дополнительной информации "При Нажатии ОК Предыдущий Плейлист будет заменен Новым.";
4. Вызвать метод setStandartButtons() для msg и передать ему список стандартных кнопок Ok и Cancel;
5. Вызвать метод setIcon () для msg и передать ему предопределенный значок Question;
6. Вызвать метод setDefaultButton() для msg и передать ему стандартную кнопку Ok для установки по умолчанию;
7. Инициализировать переменную int res и передать ей значение, возвращенное вызовом метода msg.exec();
8. Если res = QMessageBox::Ok, перейти к шагу 9, иначе к шагу 37;
9. Объявить QFile file;
10. Вызвать метод file.setFileName() и передать в него путь файла "D:/qt\_projects/music.txt";
11. Создать объект класса QFileInfo inf("D:/qt\_projects/music.txt");
12. Начать выполнение блока try;
13. Если методы inf.exists() и inf.isFile() возвращают ложное значение, перейти к шагу 14, иначе перейти к шагу 15;
14. Сформировать флаг исключения типа FileException<Widget>() и передать в него (Errors::fileIsNotOpen, this, "Не удалось открыть файл") и перейти к шагу 24;
15. Если вызов метода file.open(QIODevice::WriteOnly) вернул истинное значение, перейти к шагу 16, иначе перейти к шагу 23;
16. Объявить объект QTextStream stream(&file);
17. Инициализировать объект класса MyIterator<QString> iter и передать ему значение, возвращенное вызовом метода list.begin();
18. Начать цикл int i = 0, i < list.getsize(), ++i;
19. Использовать перегруженный оператор << для записи в stream \*iter и ‘\n’;
20. Использовать перегруженный оператор ++iter для присваивания iter ссылки на следующий объект;
21. Если i < list.getsize(), перейти к шагу 19, иначе к шагу 22;
22. Закрыть файл, вызвав метод file.close(),
23. Завершить выполнение блока try и перейти к шагу 37;
24. Начать выполнение блока catch() и передать ему объект, хранящий информацию исключения FileException<Widget> obj;
25. Вызвать метод отображения исключения obj.show();
26. Вызвать метод setText() для msg и передать ему текст основного сообщения "Хотите Создать файл сохранения Плейлиста?";
27. Вызвать метод setInformativeText() для msg и передать ему текст дополнительной информации "При Нажатии ОК будет создан новый файл плейлиста по адресу D:/qt\_projects/music.txt.";
28. Вызвать метод setStandartButtons() для msg и передать ему список стандартных кнопок Ok и Cancel;
29. Вызвать метод setIcon () для msg и передать ему предопределенный значок Question;
30. Вызвать метод setDefaultButton() для msg и передать ему стандартную кнопку Ok для установки по умолчанию;
31. Передать переменной int res значение, возвращенное вызовом метода msg.exec();
32. Если res = QMessageBox::Ok, перейти к шагу 9, иначе к шагу 37;
33. Инициализировать QFile file("D:/qt\_projects/music.txt");
34. Вызвать метод file.open(QIODevice::WriteOnly);
35. Закрыть файл, вызвав метод file.close() ;
36. Завершить блок catch() и перейти к шагу 37;
37. Конец.

## 3.3 Обработчик void Widget::on\_Music\_clicked()

1. Объявить список строк nameFilters типа QStringList;
2. Применить оператор << и передать списку nameFilters строку "\*mp3";
3. Создать объект, хранящий путь к директории Music QDir dirFrom("C:/Users/shala/Music/");
4. Инициализировать список строк fileList типа QStringList и присвоить ему значение возвращенное методом dirFrom.entryList(nameFilters,QDir::Files);
5. Вызвать конструктор списка, хранящий объекты типа QFileInfo и передать ему значение, возвращенное вызовом метода listinfo(dirFrom.entryInfoList(nameFilters,QDir::Files);
6. Начать цикл int i = 0, i < listinfo.size(), ++i;
7. Инициализировать объект s типа QString и присвоить ему значение, возвращенное последовательностью вызовов методов listinfo.at(i).absoluteFilePath();
8. Вызвать метод добавления в конец списка и передать ему строку s list.push(s);
9. Если i < listinfo.size(), перейти к шагу 7, иначе к шагу 10;
10. Начать блок макроса foreach(){} переработать элементы списка строк fileList в единую строку filePath типа QString;
11. Объявить список items типа QList с объектами типа QStandartItem\*;
12. Вызовом метода append() поместить в список items значения типа new QStandartItem(QDir(filePath).dirName());
13. Вызовом метода append() поместить в список items значения типа new QStandartItem("C:/Users/shala/Music/"+filePath);
14. Вызовом метода appendRow(), передав ему список items, установить список в модель данных плейлиста для отображения m\_playListModel;
15. Вызовом метода addMedia(), передав ему значения QUrl("C:/Users/shala/Music/"+filePath), установить URL-адреса плейлист проигрывателя m\_playlist;
16. Завершить макрос foreach(){};
17. Конец.

## 3.4 Обработчик void Widget::on\_Lastplaylist\_clicked()

1. Объявить QFile file;
2. Вызвать метод установки имени файла file.setFileName() и передать в него путь файла "D:/qt\_projects/music.txt";
3. Вызвать конструктор класса QFileInfo и создать объект inf("D:/qt\_projects/music.txt");
4. Начать выполнение блока try;
5. Если методы существования файла inf.exists() и определения файла inf.isFile() возвращают ложное значение, перейти к шагу 6, иначе перейти к шагу 7;
6. Сформировать флаг исключения типа FileException<Widget>() и передать в него (Errors::fileIsNotOpen, this, "Не удалось открыть файл") и перейти к шагу catch;
7. Если вызов метода открытия файла file.open(QIODevice::ReadOnly) вернул истинное значение, перейти к шагу 8, иначе перейти к шагу 47;
8. Объявить объект QTextStream stream(&file);
9. Начать цикл пока вызов метода определения конца текста in.atEnd() возвращает ложное значение;
10. Инициализировать строку direc типа QString и присвоить ей значение возвращаемое методом чтения строки in.readLine();
11. Объявить список строк nameFilters типа QStringList;
12. Применить оператор << и передать списку nameFilters строку "\*mp3";
13. Вызвать конструктор класса QFileInfo fileinfo() и передать в него значение direc;
14. Создать объект dirpath типа QString и присвоить ему значение возвращаемое методом возврата пути файла fileinfo.absolutePath();
15. Создать объект dirnamepath типа QString и присвоить ему значение возвращаемое вызовом метода возврата имени и пути файла fileinfo.absoluteFilePath();
16. Создать объект dirname типа QString и присвоить ему значение возвращаемое вызовом метода возврата имени файла fileinfo.fileName();
17. Вызвать конструктор итератора директории QDirIterator it() и передать в него dirpath и QDir::Files;
18. Передвинуть итератор на следующий элемент методом it.next();
19. Начать цикл пока итератор указывает на существующий файл, для чего вызвать последовательность методов it.fileInfo().exists();
20. Инициализировать строку dirname2 типа QString и присвоить ей значение возвращаемое методом возврата имени файла обращаясь по итератору it.fileInfo().fileName();
21. Если dirname2 = dirname перейти к шагу 22, иначе к шагу 28;
22. Добавить в конец списка list имя и путь файла вызвав метод list.push(dirnamepath);
23. Объявить список items типа QList с объектами типа QStandartItem\*;
24. Вызовом метода append() поместить в список items значения типа new QStandartItem(dirname);
25. Вызовом метода append() поместить в список items значения типа new QStandartItem(dirnamepath);
26. Вызовом метода appendRow(), передав ему список items, установить список в модель данных плейлиста для отображения m\_playListModel;
27. Вызовом метода addMedia(), передав ему значения QUrl::fromLocalFile(it.filePath()), установить URL-адреса плейлист проигрывателя m\_playlist;
28. Перейти по итератору на следующий элемент в директории it.next();
29. Если метод типа bool it.fileInfo().exists() вернул true, перейти к шагу 20, иначе к шагу 30;
30. Если метод in.atEnd() вернул ложное значение перейти к шагу 10, иначе к шагу 31;
31. Закрыть файл вызвав метод file.close();
32. Завершить блок try и перейти к шагу 47;
33. Начать блок catch;
34. Начать выполнение блока catch() и передать ему объект, хранящий информацию исключения FileException<Widget> obj;
35. Вызвать метод отображения исключения obj.show();
36. Вызвать метод setText() для msg и передать ему текст основного сообщения "Хотите Создать файл сохранения Плейлиста?";
37. Вызвать метод setInformativeText() для msg и передать ему текст дополнительной информации "При Нажатии ОК будет создан новый файл плейлиста по адресу D:/qt\_projects/music.txt.";
38. Вызвать метод setStandartButtons() для msg и передать ему список стандартных кнопок Ok и Cancel;
39. Вызвать метод setIcon () для msg и передать ему предопределенный значок Question;
40. Вызвать метод setDefaultButton() для msg и передать ему стандартную кнопку Ok для установки по умолчанию;
41. Передать переменной int res значение, возвращенное вызовом метода msg.exec();
42. Если res = QMessageBox::Ok, перейти к шагу 9, иначе к шагу 37;
43. Инициализировать QFile file("D:/qt\_projects/music.txt");
44. Вызвать метод file.open(QIODevice::WriteOnly);
45. Закрыть файл, вызвав метод file.close() ;
46. Завершить блок catch() и перейти к шагу 37;
47. Конец.

# 4 ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ

Как было сказано ранее, этот проект создан для прослушивания аудиофайлов. Для того чтобы прослушать треки необходимо:

1. Добавить аудиофайлы, на главной панели(приложение Б.1) имеется выбор способа добавления:

* Добавить из папки Музыка на устройстве.
* Добавить с помощью выбора файлов через диалоговое окно(приложение Б.2).
* Добавить аудиофайлы из последнего сохраненного плейлиста.

1. С помощью кнопок управления треком и плейлистом(передвижение назад(вперед) по плейлисту, пауза, проигрывание, проиграть трек заново) начать прослушивание добавленного плейлиста.
2. По желанию пользователя возможно сохранение добавленного плейлиста или его очистка(приложение Б.3).
3. Если вывелась ошибка при открытии файла(приложение Б.4), хранящего аудиофайлы, рекомендуется при выводе окна вопроса выбрать вариант “Ok”(создать файл, хранящий аудиофайлы из плейлиста).

# ЗАКЛЮЧЕНИЕ

В этом курсовом проекте был реализован аудиоплеер с исползованием фреймворка Qt на основе изученного материала и знаний из дисциплины "Программирование на языках высокого уровня". Основными плюсами этой программы являются: изменение величины звука в самой программе, сохранение плейлиста и его добавление при следующем запуске, добавление треков через диалоговое окно, добавление треков из папки “музыка”.

В дальнейшем данный проект может быть улучшен в следующих аспектах:

1. Создание нескольких плейлистов.
2. Сортировка плейлистов.
3. Создание плейлиста по умолчанию, в который добавляются все файлы формата .mp3 на этом устройстве.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1.  М. Шлее, Qt 5.10. Профессиональное программирование на С++. -СПб.: БХВ-Петербург, 2018. -1072 с.: ил. — (В подлиннике) — с. 392 – 401.

2. Qt Multimedia Documentation [Электронный ресурс]. — Режим доступа: <https://doc.qt.io/qt-5/qtmultimedia-index.html>.

3. Simple and Fast Multimedia Library [Электронный ресурс]. — Режим доступа: <https://www.sfml-dev.org/tutorials/2.5/>.

4. Воспроизведение звука при помощи библиотеки BASS [Электронный ресурс]. — Режим доступа: http://www.tiflocomp.ru/games/design/sound\_games/bass.php.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода программного средства

[\\ файл](file:///\\файл) “main.cpp”

#include "widget.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

Widget w; // Объявление объекта виджета

w.show(); // Отображение объекта виджета

return a.exec();

}

\\ файл “widget.cpp”

#include "widget.h"

#include "ui\_widget.h"

#include "shablon.h"

#include "fileexception.h"

#include <QFileDialog>

#include <QDir>

#include <QTime>

#include <QFile>

#include <QDirIterator>

#include <QMessageBox>

List<QString> list;

Widget::Widget(QWidget \*parent) : // Конструктор класса Widget

QWidget(parent),

ui(new Ui::Widget)

{

ui->setupUi(this);

// Настройка таблицы плейлиста

m\_playListModel = new QStandardItemModel(this);

ui->playlistView->setModel(m\_playListModel); // Устанавливаем модель данных в TableView

// Устанавливаем заголовки таблицы

m\_playListModel->setHorizontalHeaderLabels(QStringList() << tr("Audio Track") // Установка названий столбцов в модели отображения плейлиста

<< tr("File Path"));

ui->playlistView->hideColumn(1); // Скрываем колонку, в которой хранится путь к файлу

ui->playlistView->verticalHeader()->setVisible(false); // Скрываем нумерацию строк

ui->playlistView->setSelectionBehavior(QAbstractItemView::SelectRows); // Включаем выделение строк

ui->playlistView->setSelectionMode(QAbstractItemView::SingleSelection); // Разрешаем выделять только одну строку

ui->playlistView->setEditTriggers(QAbstractItemView::NoEditTriggers); // Отключаем редактирование

// Включаем подгонку размера последней видимой колонки к ширине TableView

ui->playlistView->horizontalHeader()->setStretchLastSection(true);

m\_player = new QMediaPlayer(this); // Инициализируем плеер

m\_playlist = new QMediaPlaylist(m\_player); // Инициализируем плейлист

m\_player->setPlaylist(m\_playlist); // Устанавливаем плейлист в плеер

ui->pdiaVolume->setRange(0, 100);// Установка границ громкости звукка

int nDefaultVolume = 50; // Инициализация значения звука по умолчанию

m\_player->setVolume(nDefaultVolume); // Устанавливаем громкость воспроизведения треков

ui->pdiaVolume->setValue(nDefaultVolume); // Установка значения звука по умолчанию

m\_playlist->setPlaybackMode(QMediaPlaylist::Loop); // Устанавливаем циклический режим проигрывания плейлиста

// подключаем кнопки управления к слотам управления

// Здесь отметим, что навигация по плейлисту осуществляется именно через плейлист

// а запуск/пауза/остановка через сам плеер

connect(ui->btn\_previous, &QToolButton::clicked, m\_playlist, &QMediaPlaylist::previous);

connect(ui->btn\_next, &QToolButton::clicked, m\_playlist, &QMediaPlaylist::next);

connect(ui->btn\_play, &QToolButton::clicked, m\_player, &QMediaPlayer::play);

connect(ui->btn\_pause, &QToolButton::clicked, m\_player, &QMediaPlayer::pause);

connect(ui->btn\_stop, &QToolButton::clicked, m\_player, &QMediaPlayer::stop);

connect(ui->pdiaVolume, SIGNAL(valueChanged(int)), m\_player, SLOT (setVolume (int)));//подключение слотов круговой шкалы регулировки звука

connect(ui->m\_psldPosition, SIGNAL(sliderMoved(int) ), SLOT(slotSetMediaPosition(int)) ) ;//подключение слотов слайдера

connect(m\_player, SIGNAL(positionChanged(qint64) ), SLOT(slotSetSliderPosition(qint64)) ) ;

connect(m\_player, SIGNAL(durationChanged(qint64)), SLOT(slotSetDuration(qint64)) ) ;

// При даблклике по треку в таблице устанавливаем трек в плейлисте

connect(ui->playlistView, &QTableView::doubleClicked, [this](const QModelIndex &index){

m\_playlist->setCurrentIndex(index.row());

});

// при изменении индекса текущего трека в плейлисте, устанавливаем название файла в специальном лейбле

connect(m\_playlist, &QMediaPlaylist::currentIndexChanged, [this](int index){

ui->currentTrack->setText(m\_playListModel->data(m\_playListModel->index(index, 0)).toString());

});

}

Widget::~Widget() // Деструктор класса Widget

{

delete ui;

delete m\_playListModel;

delete m\_playlist;

delete m\_player;

}

//Slots for slider

void Widget::slotSetMediaPosition(int n) // Установка позиции текущего трека

{

m\_player->setPosition(n);

}

QString Widget::msecsToString(qint64 n) // Перевод секунд в строку

{

int nHours = (n / (60 \* 60 \* 1000));

int nМinutes = ( (n % (60 \* 60 \* 1000)) / (60 \* 1000));

int nSeconds = ( (n % (60 \* 1000)) / 1000);

return QTime(nHours, nМinutes, nSeconds).toString("hh:mm:ss");

}

void Widget::slotSetDuration(qint64 n) // Устанавливает максимальное и минимальное значение слайдера и текст отображения этих значений

{

ui->m\_psldPosition->setRange(0, n); // Установить максимальное и минимальное значение

ui->m\_Current->setText(msecsToString(0)); // Отображение минимального

ui->m\_Remain->setText(msecsToString(n)); // Отображение максимального

}

void Widget::slotSetSliderPosition(qint64 n) // Устанока значений согласно текущему прогрессу проигрывания трека

{

ui->m\_psldPosition->setValue(n); // Установить текущее значение

ui->m\_Current->setText(msecsToString(n)); // Отобразить как текущее

int nDuration = ui->m\_psldPosition->maximum(); // Присвоение переменноой максимального значения

ui->m\_Remain->setText(msecsToString(nDuration -n) ); // Отобразить оставшееся время проигрывания трека

}

//Slots for slider end

void Widget::on\_btn\_add\_clicked()

{

// С помощью диалога выбора файлов делаем множественный выбор mp3 файлов

QStringList files = QFileDialog::getOpenFileNames(this,

tr("Open files"),

QString(),

tr("Audio Files (\*.mp3)"));

for (int i = 0; i < files.size(); ++i) {

list.push(files.at(i));

}

// Далее устанавливаем данные по именам и пути к файлам

// в плейлист и таблицу отображающую плейлист

foreach (QString filePath, files) {

QList<QStandardItem \*> items;

items.append(new QStandardItem(QDir(filePath).dirName()));

items.append(new QStandardItem(filePath));

m\_playListModel->appendRow(items);

m\_playlist->addMedia(QUrl(filePath));

}

}

void Widget::on\_Save\_clicked()

{

QMessageBox msg;

msg.setText("Вы действительно хотите Сохранить Нынешний Плейлист?"); // Начало настройки диалогового окна

msg.setInformativeText("При Нажатии ОК Предыдущий Плейлист будет заменен Новым.");

msg.setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel);

msg.setIcon(QMessageBox::Question);

msg.setDefaultButton(QMessageBox::Ok); // Конец настройки диалогового окна

int res = msg.exec();

if (res == QMessageBox::Ok)

{

QFile file;

file.setFileName("D:/qt\_projects/music.txt");

QFileInfo inf("D:/qt\_projects/music.txt");

try // Блок try{} для генерации исключения при отсутствии файла

{

if (!inf.exists() && !inf.isFile())

throw FileException<Widget>(Errors::fileIsNotOpen, this, "Не удалось открыть файл");

else

{

if(file.open(QIODevice::WriteOnly)) // Запись в файл треков находящихся в плейлисте

{

QTextStream stream(&file);

MyIterator<QString> iter = list.begin();

for (int i = 0; i < list.getsize(); ++i) {

stream << \*iter << '\n';

++iter;

}

}

else {

return;

}

file.close();

}

}

catch(FileException<Widget> obj) // Блок catch(){} для обработки исключения, вывода окна ошибки и ее исправление

{

obj.show();

msg.setText("Хотите Создать файл сохранения Плейлиста?");

msg.setInformativeText("При Нажатии ОК будет создан новый файл плейлиста по адресу D:/qt\_projects/music.txt.");

msg.setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel);

msg.setIcon(QMessageBox::Question);

msg.setDefaultButton(QMessageBox::Ok);

res = msg.exec();

if (res == QMessageBox::Ok) // Создание файла при выборе Ok

{

QFile file("D:/qt\_projects/music.txt");

file.open(QIODevice::WriteOnly);

file.close();

}

else

return;

return;

}

}

else

return;

return;

}

void Widget::setting()

{

m\_playListModel->clear(); // Очистка модели отображения плейлиста

m\_playlist->clear(); // Очистка плейлиста

// Настройка отбражения плейлиста

m\_playListModel->setHorizontalHeaderLabels(QStringList() << tr("Audio Track")

<< tr("File Path"));

ui->playlistView->hideColumn(1); // Скрываем колонку, в которой хранится путь к файлу

ui->playlistView->verticalHeader()->setVisible(false); // Скрываем нумерацию строк

ui->playlistView->setSelectionBehavior(QAbstractItemView::SelectRows); // Включаем выделение строк

ui->playlistView->setSelectionMode(QAbstractItemView::SingleSelection); // Разрешаем выделять только одну строку

ui->playlistView->setEditTriggers(QAbstractItemView::NoEditTriggers); // Отключаем редактирование

// Включаем подгонку размера последней видимой колонки к ширине TableView

ui->playlistView->horizontalHeader()->setStretchLastSection(true);

}

void Widget::on\_Clear\_clicked() // Функция очистки плйлиста

{

QMessageBox msg;

msg.setText("Вы действительно хотите Очистить Нынешний Плейлист?");

msg.setInformativeText("При Нажатии ОК Плейлист будет Очищен.");

msg.setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel);

msg.setIcon(QMessageBox::Question);

msg.setDefaultButton(QMessageBox::Ok);

int res = msg.exec();

if (res == QMessageBox::Ok)

{

list.DeleteAll();

setting();

}

else

return;

}

void Widget::on\_Music\_clicked() // Функция добавления аудиофайлов из папки Музыка

{

QStringList nameFilters;

nameFilters<<"\*mp3";

QDir dirFrom("C:/Users/shala/Music/");

QStringList fileList = dirFrom.entryList(nameFilters,QDir::Files);

QList<QFileInfo> listinfo(dirFrom.entryInfoList(nameFilters,QDir::Files));

for (int i = 0; i < listinfo.size(); ++i) {

QString s = listinfo.at(i).absoluteFilePath();

list.push(s);

}

foreach (QString filePath, fileList)

{

QList<QStandardItem \*> items;

items.append(new QStandardItem(QDir(filePath).dirName()));

items.append(new QStandardItem("C:/Users/shala/Music/"+filePath));

m\_playListModel->appendRow(items);

m\_playlist->addMedia(QUrl("C:/Users/shala/Music/"+filePath));

}

}

void Widget::on\_Lastplaylist\_clicked() // Функция добавления аудиофайлов из последнего сохраненного плейлиста

{

QFile file;

file.setFileName("D:/qt\_projects/music.txt");

QFileInfo inf("D:/qt\_projects/music.txt");

try // Блок try{} для генерации исключения при отсутствии файла

{

if (!inf.exists() && !inf.isFile())

throw FileException<Widget>(Errors::fileIsNotOpen, this, "Не удалось открыть файл");

else

{

if(file.open(QIODevice::ReadOnly)) // Блок вывода треков из файла в плейлист

{

QTextStream in(&file);

while(!in.atEnd())

{

QString direc = in.readLine();

QStringList nameFilters;

nameFilters<<"\*mp3";

QFileInfo fileinfo(direc);

QString dirpath = fileinfo.absolutePath();

QString dirnamepath = fileinfo.absoluteFilePath();

QString dirname = fileinfo.fileName();

QDirIterator it(dirpath, QDir::Files);

it.next();

while (it.fileInfo().exists())

{

QString dirname2 = it.fileInfo().fileName();

if(dirname2 == dirname)

{

list.push(dirnamepath);

QList<QStandardItem \*> items;

items.append(new QStandardItem(dirname));

items.append(new QStandardItem(dirnamepath));

m\_playListModel->appendRow(items);

m\_playlist->addMedia(QUrl::fromLocalFile(it.filePath()));

}

it.next();

}

}

}

else

return;

file.close();

}

}

catch(FileException<Widget> obj) // Блок catch(){} для обработки исключения, вывода окна ошибки и ее исправление

{

obj.show();

QMessageBox msg;

msg.setText("Хотите Создать файл сохранения Плейлиста?");

msg.setInformativeText("При Нажатии ОК будет создан новый файл плейлиста по адресу D:/qt\_projects/music.txt.");

msg.setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel);

msg.setIcon(QMessageBox::Question);

msg.setDefaultButton(QMessageBox::Ok);

int res = msg.exec();

if (res == QMessageBox::Ok) // Создание файла при выборе Ok

{

QFile file("D:/qt\_projects/music.txt");

file.open(QIODevice::WriteOnly);

file.close();

}

else

return;

return;

}

return;

}

\\ файл “widget.h”

#ifndef WIDGET\_H

#define WIDGET\_H

#include <QWidget>

#include <QStandardItemModel>

#include <QMediaPlayer>

#include <QMediaPlaylist>

namespace Ui {

class Widget;

}

class Widget : public QWidget

{

Q\_OBJECT

public:

explicit Widget(QWidget \*parent = 0);

QString msecsToString(qint64 n);// Вспомогательная функция слайдера(запись времени трека в строку)

void setting(); // Очистка и настройка отображения плейлиста

~Widget();

private slots:

void on\_btn\_add\_clicked();// Слот для обработки добавления треков через диалоговое окно

void on\_Save\_clicked();// Слот для сохранения сформированного плейлиста

void on\_Clear\_clicked();// ССлот для очистки сформированного плейлиста

void on\_Lastplaylist\_clicked();// Слот для обработки добавления треков из последнего сохраненного плейлиста

void on\_Music\_clicked();// Слот для обработки добавления треков из папки Music

void slotSetSliderPosition(qint64 ) ;// Слоты слайдера

void slotSetMediaPosition (int );

void slotSetDuration (qint64 ) ;

private:

Ui::Widget \*ui;

QStandardItemModel \*m\_playListModel; // Модель данных плейлиста для отображения

QMediaPlayer \*m\_player; // Проигрыватель треков

QMediaPlaylist \*m\_playlist; // Плейлиста проигрывателя

};

#endif // WIDGET\_H

\\ файл “shablon.h”

#ifndef LIST\_H

#define LIST\_H

template <class T>

struct Node

{

T info;

Node<T>\* next = nullptr;

Node<T>\* prev = nullptr;

};

template <class T>

class MyIterator

{

Node<T>\* mPointer;

int index = 0;

public:

MyIterator (Node<T>\* pointer);

void setPointer(Node<T> \*newPointer);

Node<T>\* getPointer();

T& operator\*() const;

MyIterator& operator++();

MyIterator& operator++(int);

MyIterator& operator--();

T& operator[](int step);

bool operator==(const MyIterator& secondParam) const;

bool operator!=(const MyIterator& secondParam) const;

bool operator<(const MyIterator& secondParam) const;

bool operator>=(const MyIterator& secondParam) const;

friend MyIterator<T> operator+(MyIterator<T> iter, int step)

{

for (int i = 0; i < step; i++)

++iter;

return iter;

}

};

template <class T>

class List

{

Node<T>\* head = nullptr;

Node<T>\* tail = nullptr;

int size=0;

public:

void DeleteAll();

void push(T element);

void deleteElement(int index);

int getsize();

MyIterator<T> begin();

MyIterator<T> end();

friend class MyIterator<T>;

};

template<class T>

Node<T>\* MyIterator<T>::getPointer()

{

return mPointer;

}

template <class T>

MyIterator<T>::MyIterator(Node<T>\* pointer): mPointer(pointer){};

template <class T>

T& MyIterator<T>::operator\*() const

{

return mPointer->info;

}

template <class T>

MyIterator<T>& MyIterator<T>::operator++()

{

mPointer=mPointer->next;

return \*this;

}

template <class T>

MyIterator<T>& MyIterator<T>::operator++(int)

{

MyIterator temp = \*this;

mPointer = mPointer->next;

return temp;

}

template <class T>

MyIterator<T>& MyIterator<T>::operator--()

{

mPointer = mPointer->prev;

return \*this;

}

template<class T>

T &MyIterator<T>::operator[](int step)

{

Node<T>\* pointer = mPointer;

for (int i = 0; i < step; i++)

pointer = pointer->next;

return pointer->info;

}

template <class T>

bool MyIterator<T>::operator==(const MyIterator& secondParam) const

{

return mPointer == secondParam.mPointer;

}

template <class T>

bool MyIterator<T>::operator!=(const MyIterator& secondParam) const

{

return mPointer != secondParam.mPointer;

}

template <class T>

bool MyIterator<T>::operator<(const MyIterator& secondParam) const

{

return mPointer->info < secondParam.mPointer->info;

}

template <class T>

bool MyIterator<T>::operator>=(const MyIterator& secondParam) const

{

return index >= secondParam.index;

}

/////////////////////////////////////////////////

template<class T>

void List<T>::DeleteAll()

{

Node<T>\* tmp = tail;

for (size\_t i = size; i > 0; i--)

{

if (i == 1)

{

head = tail = nullptr;

size=0;

}

else

{

Node<T>\* tmp = tail;

tail = tail->prev;

tail->next = nullptr;

delete tmp;

}

}

}

template<class T>

void List<T>::push(T element)

{

Node<T>\* elem = new Node<T>;

elem->info = element;

if (!head)

{

head = tail = elem;

head->next = tail->next;

size++;

}

else

{

tail->next = elem;

elem->prev = tail;

tail = elem;

size++;

}

}

template<class T>

int List<T>::getsize()

{

int s = size;

return s;

}

template<class T>

void List<T>::deleteElement(int index)

{

if (!head)

return;

Node<T>\* temp;

MyIterator<T> it = head;

for (int i = 0; i < index - 1; it++, i++);

temp = it.getPointer();

if (index > 0)

{

Node<T>\* temp2 = temp->next;

if (temp2 == tail)

tail = 0;

else

temp2->next->prev = temp;

temp->next = temp->next->next;

delete temp2;

}

else

{

if (head == tail)

tail = head = nullptr;

else

{

head = head->next;

head->prev = nullptr;

}

delete temp;

}

}

template<class T>

MyIterator<T> List<T>::begin()

{

return MyIterator<T>(head);

}

template<class T>

MyIterator<T> List<T>::end()

{

return MyIterator<T>(tail);

}

#endif // LIST\_H

\\ файл “exeption.h”

#ifndef EXCEPTION\_H

#define EXCEPTION\_H

#include <QString>

#include <QMessageBox>

enum class Errors

{

emptyLines = 100,

equalLines = 101,

fileIsNotOpen = 102,

incorrectInput = 103,

};

class Exception

{

protected:

Errors code;

QString message;

virtual void show() = 0;

};

#endif // EXCEPTION\_H

\\ файл “fileexeption.h”

#ifndef FILEEXCEPTION\_H

#define FILEEXCEPTION\_H

#include "exception.h"

template <class T>

class FileException :

public Exception

{

T\* window;

public:

FileException<T>(Errors code, T\* window, QString message);

void show() override;

};

template <class T>

FileException<T>::FileException(Errors code, T\* window, QString message)

{

this->code = code;

this->window = window;

this->message = message;

}

template <class T>

void FileException<T>::show()

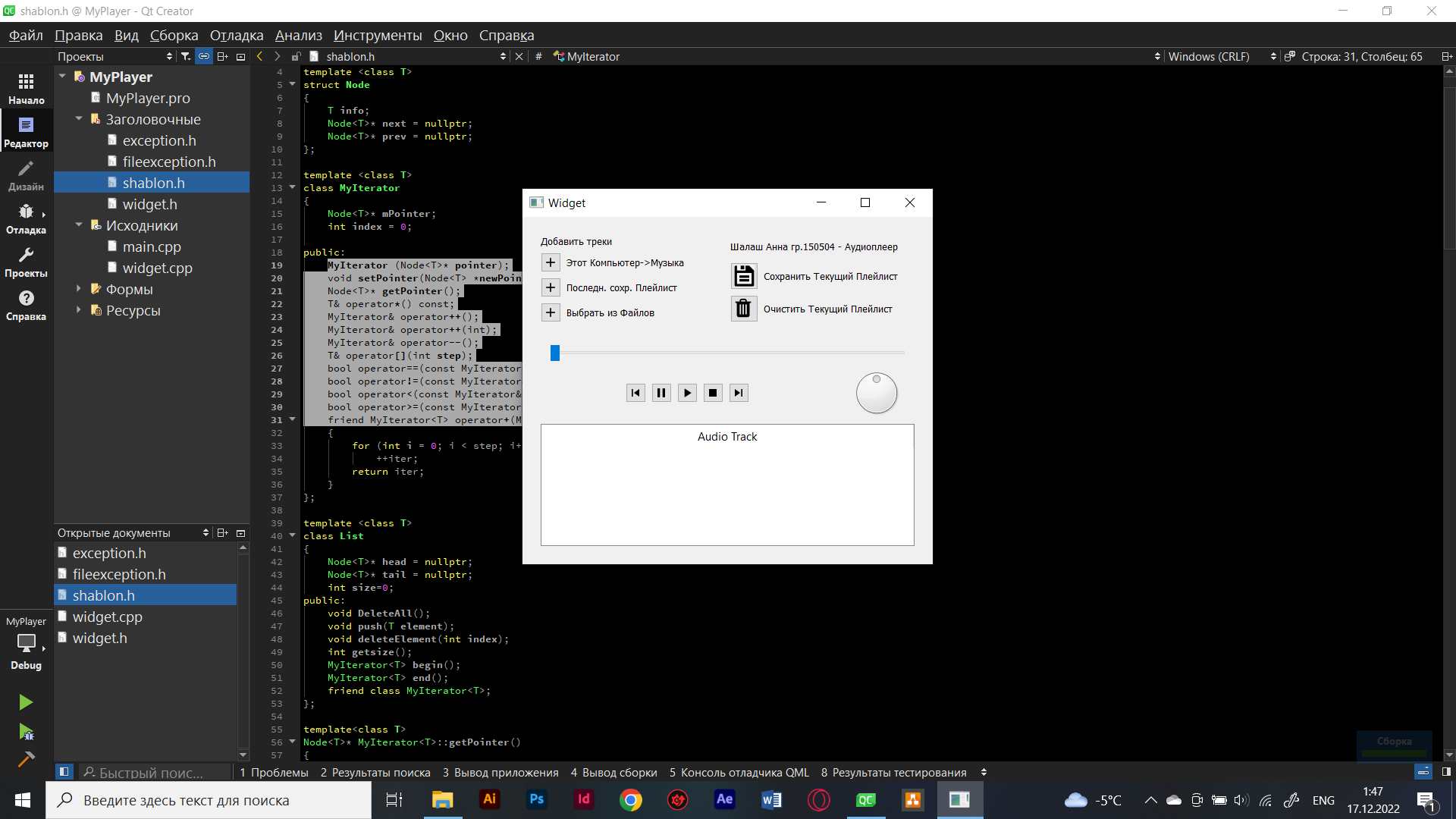
{

QMessageBox::warning(window, "Ошибка " + QString::number((int)code), message);

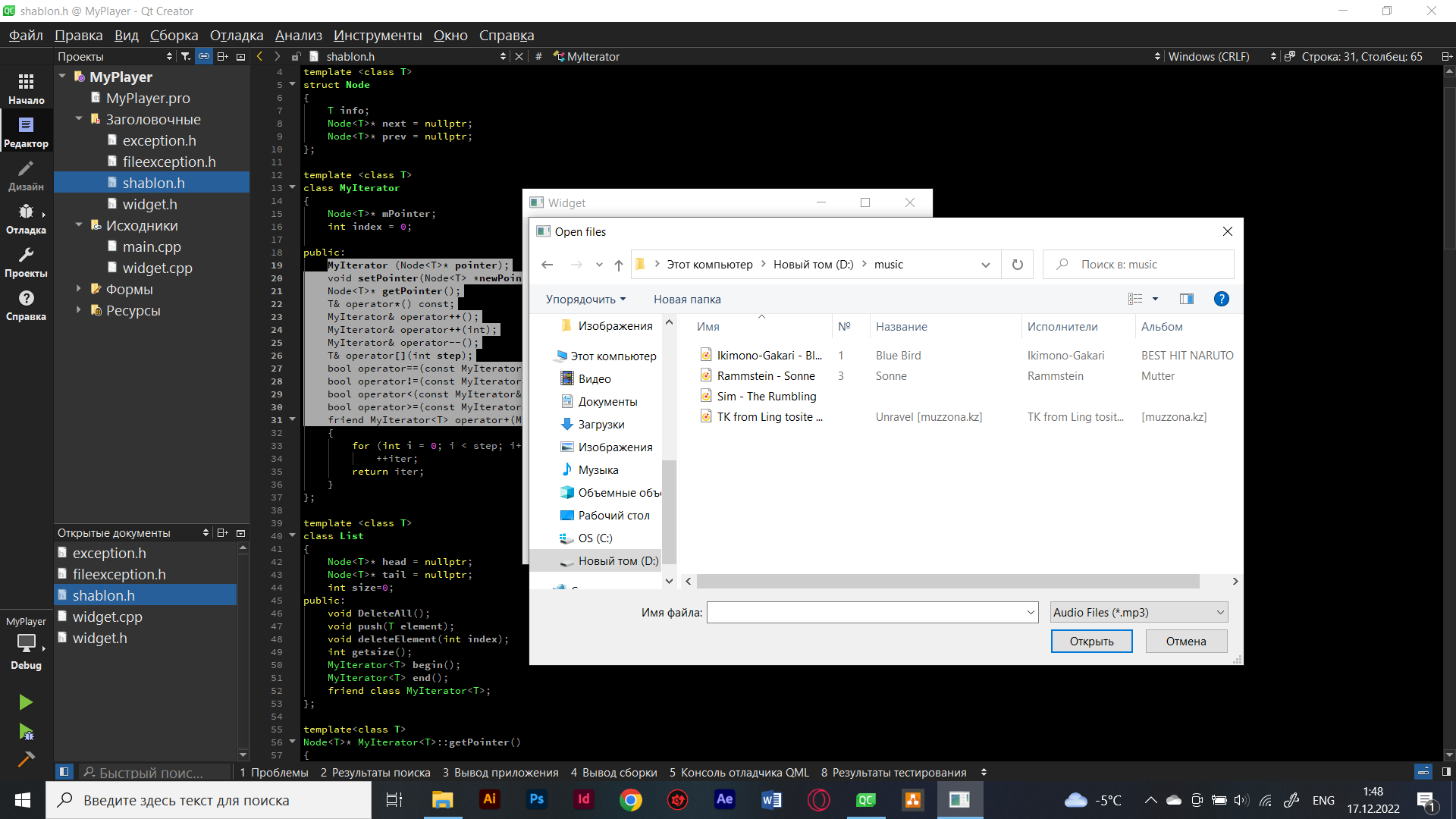
}

#endif // FILEEXCEPTION\_H

# ПРИЛОЖЕНИЕ Б (обязательное) Скриншоты работы программы

****

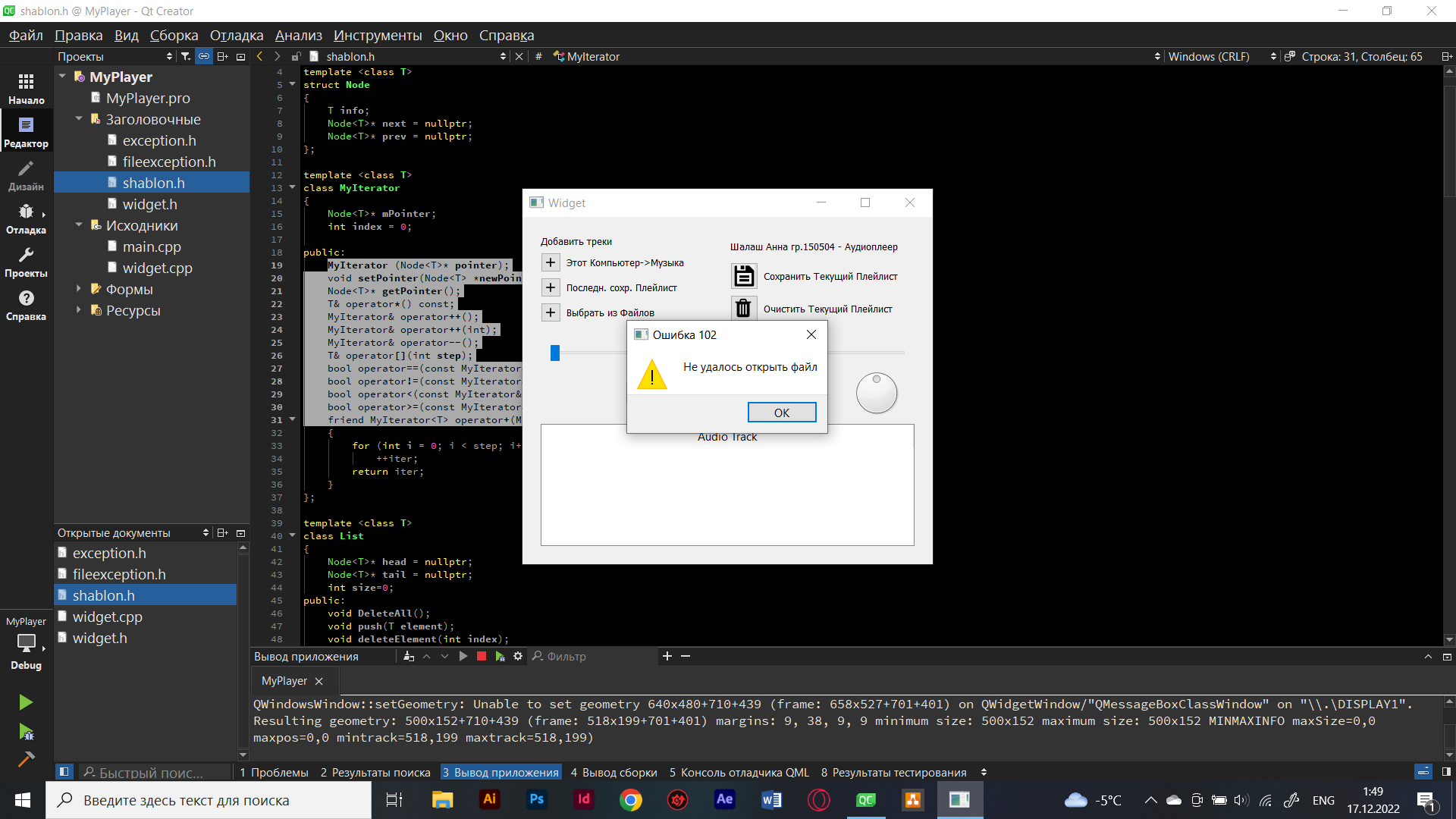
Б.1 — Главная панель аудиоплеера



Б.2 — Диалоговое окно для выбора аудиофайлов

|  |  |
| --- | --- |
| C:\Users\shala\OneDrive\Изображения\Screenshots\2022-12-17 (2).png | C:\Users\shala\OneDrive\Изображения\Screenshots\2022-12-17 (3).png |

Б.3 — Окна вопросов для выбора действия



Б.4 — Окно вывода ошибки открытия файла при обработке исключения

# ПРИЛОЖЕНИЕ В (обязательное) UML-диаграмма классов программного средства

# ПРИЛОЖЕНИЕ Г (обязательное) Блок-схемы алгоритмов функций on\_Clear\_clicked(), DeleteAll(), push()

# ПРИЛОЖЕНИЕ Д (обязательное) Ведомость документов