СОДЕРЖАНИЕ

[ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ 5](#_Toc122038307)

[ВВЕДЕНИЕ 6](#_Toc122038308)

[1 ОБЗОР ЛИТЕРАТУРЫ 7](#_Toc122038309)

[2 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ 8](#_Toc122038310)

[3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 15](#_Toc122038311)

[4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 18](#_Toc122038312)

[ЗАКЛЮЧЕНИЕ 21](#_Toc122038313)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 22](#_Toc122038314)

[ПРИЛОЖЕНИЕ А 23](#_Toc122038315)

[ПРИЛОЖЕНИЕ Б 40](#_Toc122038316)

[ПРИЛОЖЕНИЕ В 47](#_Toc122038317)

[ПРИЛОЖЕНИЕ Г 48](#_Toc122038318)

[ПРИЛОЖЕНИЕ Д 49](#_Toc122038319)

# ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ

Задание на курсовой проект – программа для чтения книг. Такое приложение должно быть простым в использовании, обладать удобным пользовательским интерфейсом и минимально необходимым набором функций, позволяющим читать электронные книги. К таким функциям можно отнести открытие, просмотр и перемещение по страницам книги. Также необходимо реализовать возможность открывать файлы после повторного запуска приложения на последней открытой странице.

# ВВЕДЕНИЕ

Чтение книг всегда являлось важной части жизни каждое человека независимо от рода его деятельности. В 21 веке все большую и большую популярность приобретает чтение книг в онлайн формате на телефоне или компьютере. В связи с этим было решено попробовать создать свое приложение для чтения книг.

Объектно-ориентированное программирование представляет собой концепцию программирования, в которой программа представляет из себя совокупность взаимодействующих объектов, каждый из которых является экземпляром какого-либо класса, а сами классы образуют иерархию наследования. Один из наиболее популярных языков программирования, поддерживающим данную концепцию, является C++.

C++ — компилируемый статически типизированный язык программирования. Он поддерживает такие парадигмы программирования, как процедурное, объектно-ориентированное и обобщенное. C++ включает в себя как возможности низкоуровневых, так и высокоуровневых языков программирования.

Основными принципами объектно-ориентированного программирования являются инкапсуляция, наследование и полиморфизм. C++ предоставляет возможности для реализации данных концепций. Инкапсуляция позволяет ограничить доступ к свойствам объектов другим частям этой же программы. Наследование предоставляет возможность повторного использования кода, что помогает уменьшить количество кода и увеличить скорость его написания. Полиморфизм позволяет быть программе более гибкой.

# 1 ОБЗОР ЛИТЕРАТУРЫ

В наше время существует множество различных форматов для чтения книг, например fb2 (fb – fiction book, “художественная книга”, epub (electronic publication, “электронная публикация”), pdf, mobi. Каждый из этих форматов обладает своими достоинствами и недостатками. Так, основным достоинством формата epub является гибкость. Этот формат позволяет создавать динамическую разметку документа, адаптирующуюся под экраны различных устройств. Недостатком же является сложность обработки файлов такого формата. Основным достоинством и недостатком формата fb2 является его простота. Формат fb2 – это частный случай xml-документа, в котором данные хранятся в тегах, что упрощает его обработку. Однако данный формат не предусматривает хранение текста научной литературы, о чем и говорит название формата. Исходя из этого было решено создать программу для чтения книг в формате fb2.

Для написания такой программы был выбран фреймворк Qt, так как он обладает большим количеством классов, которые могут пригодиться в процессе разработки программы, начиная от элементов графического интерфейса и заканчивая классами для работы с xml-документами.

Для написания логики приложения был выбран язык программирования C++, так как он обладает высокой скоростью и поддерживает парадигму объектно-ориентированного программирования.

# 2 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе приведено описание классов, созданных в процессе разработки программы для чтения книг.

**2.1 MainWindow**

Данный класс наследуется от класса QMainWindow. Данный класс является графическим представлением окна, в котором проходит работа приложения. В конструкторе вызываются методы, отвечающие за внешний вид окна.

Поля:

- Ui::MainWindow\* ui – указатель, позволяющий обращаться к виджетам главного окна;

- Book book – книга, выбранная пользователем;

- const int WINDOW\_WIDTH = 1360 – ширина окна;

- const int WINDOW\_HEIGHT = 800 – высота окна.

Методы:

- void on\_aboutProgramButton\_clicked() – срабатывает при нажатии кнопки “О программе”, открывает меню с описанием приложения;

- void on\_toStartMenuButton\_clicked() – срабатывает при нажатии кнопки “На главную”, открывает главное меню приложения;

- void on\_chooseFileButton\_clicked() – срабатывает при нажатии кнопки “Выбрать файл”, открывает меню выбора файла;

- void on\_nextPageButton\_clicked() – срабатывает при нажатии кнопки

“->”, переходит, если возможно, на следующую страницу книги;

- void on\_previousPageButton\_clicked() – срабатывает при нажатии кнопки “<-”, переходит, если возможно, на предыдущую страницу книги;

- void on\_findPageButton\_clicked() – срабатывает при нажатии кнопки с изображенной на ней лупой, переходит, если возможно, на заданную страницу;

- void on\_chooseRecentOpenedFileButton\_clicked() – срабатывает при нажатии кнопки “Выбрать ранее открытый файл”, открывает диалоговое окно для выбора файла;

- void ConfigureMainWindow() – задает стили главного окна;

- void ConfigureTabWidget() – задает стили правой части окна, где отображается различная информация;

- void ConfigureMainWindowLabel() – задает стили главному меню;

- void ConfigureAboutProgramLabel() – задает стили меню с описанием программы;

- void ConfigureBookTab() – задает стили меню с текстом книги;

- void ConfigurePageButtons() – задать стили кнопкам для переключение страниц книги;

- void SetBookLabelText(List<QString> bookText, int currentPageNumber) – устанавливает текст книги;

- void AppendPageNumberToBookFile(int pageNumber) – сохраняет номер страницы в файл.

**2.2 Book.h**

Данный класс хранит в себе все данные о книге, выбранной пользователем.

Поля:

- QString pathToBookFile – путь до выбранного файла;

- List<QString> bookText – список строк книги;

- const int MAX\_STRING\_LENGTH = 95 – максимальная длина строки;

- const int MAX\_STRING\_AMOUNT = 29 – максимальное число строк на странице;

- int totalPagesNumber – общее число страниц в книге;

- int currentPageNumber – номер текущей страницы.

Методы:

- void SetPathToBookFile(QString \_pathToBookFile) – устанавливает путь до выбранной книги;

- QString GetPathToBookFile() const – возвращает путь до выбранной книги;

- void ParseBookFile() – обрабатывает файл книги;

- List<QString> GetBookText() const – возвращает текст книги;

- void ResetBookText() – очищает текст книги;

- int GetTotalPagesNumber() const – возвращает общее число страниц;

- void SetCurrentPageNumber(int \_currentPageNumber) – задает новый номер текущей страницы;

- int GetCurrentPageNumber() const – возвращает номер текущей страницы;

- int GetMaxStringAmount() const – получает максимальное число строк на странице;

- Book& operator=(const Book&) – перегруженный оператор присваивания;

- void SetBookText(QXmlStreamReader& xmlFile) – задать текст книги;

- void SetTitleStyle(QXmlStreamReader& xmlFile) – задает заголовку книги жирный шрифт по центру экрана;

- void SetParagraphStyle(QString paragraphText) – разбивает считанный текст абзаца на строки и задает им шрифт и отступ первой строки абзаца;

- void SetTotalPagesNumber() – считает общее число страниц.

**2.3 RecentOpenedFilesWindow**

Данный класс наследуется от класса QDialog. Данный класс является графическим представлением окна, в котором пользователь может выбрать один из ранее открытых файлов. В конструкторе вызываются методы, отвечающие за создание кнопок на окне.

Поля:

- Ui::RecentOpenedFilesWindow \*ui – указатель, позволяющий обращаться к виджетам диалогового окна;

- const int WINDOW\_WIDTH = 800 – ширина окна;

- const int WINDOW\_HEIGHT = 600 – высота окна;

- QList<QString> recentOpenedFiles – список имен ранее открытых файлов;

- QList<QPushButton\*> recentOpenedFilesButtons – список кнопок для выбора ранее открытого файла;

- QList<QPushButton\*> deleteRecentOpenedFiles – список кнопок для удаления ранее открытых файлов;

- Book book – выбранная книга.

Методы:

- Book GetBook() const – возвращает выбранную книгу;

- void RecentOpenedFileButtonClicked() – срабатывает при выборе книги из списка ранее открытых файлов;

- void DeleteRecentOpenedFileButtonClicked() – срабатывает при нажатии на кнопку с изображенной на ней урной, удаляет соответствующую ей книгу;

- void ConfigureDialogWindow() – задает стили диалогового окна;

- void SetRecentOpenedFiles() – задает имена ранее открытых файлов;

- void SetLastOpenedPageNumber(QString fileName) – задает номер последней открытой страницы выбранного файла;

- void CreateRecentOpenedFilesButtons() – создает кнопки для выбора ранее открытой книги;

- void CreateDeleteRecentOpenedFilesButtons() – создает кнопку для удаления ранее открытых книг;

- void ChangeRecentOpenedFilesButtonsPosition(int i) – сдвигает кнопки ранее открытых файлов вверх после удаления книги.

**2.4 WidgetStyle**

Данный класс возвращает стили для виджетов на окнах. Поля отсутствуют.

Методы:

- static QString GetTabStyle() – возвращает стили для QTabWidget на главном окне;

- static QString GetAboutPogramLabelStyle() – возвращает стили для меню “О программе”;

- static QString GetMainWindowLabelStyle() – возвращает стили для главного меню;

- static QString GetPageNumberLabelsStyle() – возвращает стили для виджетов с номером текущей страницы и общим числом страниц;

- static QString GetBookLabelStyle() – возвращает стили для меню с книгой;

- static QString GetFindPageButtonStyle() – возвращает стили для кнопки поиска по номеру страницы;

- static QString GetFindPageLineEditStyle() – возвращает стили для меню ввода номера страницы;

- static QString GetRecentOpenedFilesButtonsStyle() – возвращает стили для кнопок выбора ранее открытых файлов;

- static QString GetDeleteRecentOpenedFilesButtonsStyle() – возвращает стили для кнопок удаления ранее открытых файлов;

- static QString GetMainWindowButtonsStyle() – возвращает стили для кнопок на главном окне.

**2.5 template<class T> Node**

Данный класс является элементом списка. В конструкторе инициализируется значение data.

Поля:

- T data – данные в “узле” списка;

- Node<T>\* next – указатель на следующий элемент списка.

**2.6 template<class T> List**

Данный класс представляет из себя контейнер list. В конструкторе значение всех полей класса обнуляется.

Поля:

- Node<T>\* head – начало списка;

- Node<T>\* tail – конец списка;

- int size – размер списка.

Методы:

- void Append(T data) – добавление элемента в конец списка;

- void Pop() – удаление элемента из конца списка;

- void Clear() – очистка списка;

- T operator[](int index) – перегруженный оператор индексирование. Возвращает данные по заданному индексу;

- int GetSize() const – возвращает размер списка.

**2.7 template<class T> Iterator**

Данный класс является дружественным классу List. Позволяет удобно перемещаться по списку. В конструкторе задается значение currentNode.

Поля:

- Node<T>\* currentNode – текущий “узел” списка.

Методы:

- Iterator<T>& operator++() – перегруженный оператор ++, перемещающий указатель currentNode на следующую позицию;

- T& operator\*() const – перегруженный оператор \*, возвращающий данные на текущем “узле” списка.

**2.8 Exception**

Данный класс является базовым классов исключений. В конструткоре задается значение errorMessage.

Поля:

- QString errorMessage – сообщение ошибки.

Методы:

- QString GetErrorMessage() const – возвращает сообщение ошибки.

**2.9 ArgumentNullException**

Данный класс наследуется от класса Exception. Используется для обработки исключительных, когда какое-либо аргумент является пустым (например, имя файла). Поля и методы отсутствуют, в конструкторе вызывается конструктор класса Exception.

**2.10 OutOfRangeException**

Данный класс наследуется от класса Exception. Используется для обработки исключительных, когда какое-либо аргумент выходит за допустимый диапазон (например, номер страницы). Поля и методы отсутствуют, в конструкторе вызывается конструктор класса Exception.

Диаграмма классов приведена в приложении В.

# 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

**3.1 Алгоритм по шагам метода void Book::ParseBookFile()**

Шаг 1. Создать объект класса QFile bookFile, в конструктор передать pathToBookFile.

Шаг 2. Если bookFile не открылся в режиме для чтения, перейти к шагу 3, иначе перейти к шагу 5.

Шаг 3. Создать окно с ошибкой с сообщением “Что-то пошло не так! Файл невозможно открыть!”.

Шаг 4. Конец.

Шаг 5. Создать объект класса QXmlStreamReader xmlFile, в конструктор передать ссылку на bookFile.

Шаг 6. Если xmlFile не на конце перейти к шагу 7, иначе перейти к шагу 14.

Шаг 7. Считать следующий тег из xmlFile.

Шаг 8. Если считанный тег является закрывающим перейти к шагу 6, иначе перейти к шагу 9.

Шаг 9. Создать объект класса QString tag, равный тексту считанного тега.

Шаг 10. Если tag равен “body” перейти к шагу 11, иначе перейте к шагу 13.

Шаг 11. Вызвать метод SetBookText, передать в него xmlFile.

Шаг 12. Перейти к шагу 14.

Шаг 13. Перейти к шагу 6.

Шаг 14. Закрытие файла с книги.

Шаг 15. Конец.

**3.2 Алгоритм по шагам метода void Book::SetBookText**

**(QXmlStreamReader& xmlFile)**

Шаг 1. Создать объект класса Qstring tag.

Шаг 2. Если tag не равен body перейти к шагу 3, иначе перйти к шагу 14.

Шаг 3. Считать следующий тег из xmlFile.

Шаг 4. Переменная tag равна считанному тегу.

Шаг 5. Если xmlFile.isEndElement() перейте к шагу 2, иначе перейти к шагу 6.

Шаг 6. Если tag не равен "p" и tag не равен "title" перейти к шагу 7, иначе перейти к шагу 10.

Шаг 7. Считать следующий тег из xmlFile.

Шаг 8. tag = xmlFile.name().toString().

Шаг 9. Перейти к шагу 6.

Шаг 10. Если tag == "p" перейти к шагу 11, иначе перейти к шагу 12.

Шаг 11. Вызвать метод SetParagraphStyle, передать в него текст, считанный из тега <p>.

Шаг 12. Вызвать метод SetTitleStyle, принимающего параметр xmlFile.

Шаг 13. Перейти к шагу 2.

Шаг 14. Вызвать метод SetTotalPagesNumber().

Шаг 15. Конец.

**3.3 Алгоритм по шагам метода void MainWindow::SetBookLabelText**

**(List<QString> bookText, int currentPageNumber)**

Шаг 1. Создать объект класса QString pageText.

Шаг 2. Переменная int maxStringAmount равна возвращаемому значению метода book.GetMaxStringAmount().

Шаг 3. Цикл по i = 0 пока i < maxStringAmount с шагом 1.

Шаг 4. Переменная int index равна значению (currentPageNumber - 1) \* maxStringAmount + i.

Шаг 5. Если index больше bookText.size() – 1 перейти к шагу 7.

Шаг 6. Вызвать метод pageText.Append, передать в него bookText[index].

Шаг 7. Вызвать метод ui->bookLabel->setText, передать в него pageText.

Шаг 8. Конец.

**3.4 Блок-схема метода void Book::SetTitleStyle(QXmlStreamReader& xmlFile)**

Данный метод предназначен для того, чтобы установить различные заголовки книги по центру жирным шрифтом.

Блок-схема данного метода приведена в приложении Г на чертеже ГУИР.400201.418 Г.1

**3.5 Блок-схема метода void MainWindow::SetBookLabelText**

**(List<QString> bookText, int currentPageNumber)**

Данный метод предназначен для отображения текста страницы на окне.

Блок-схема данного метода приведена в приложении Г на чертеже ГУИР.400201.418 Г.2.

# 4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для запуска приложения необходимо запустить файл BookReader.exe (рис. 4.1).

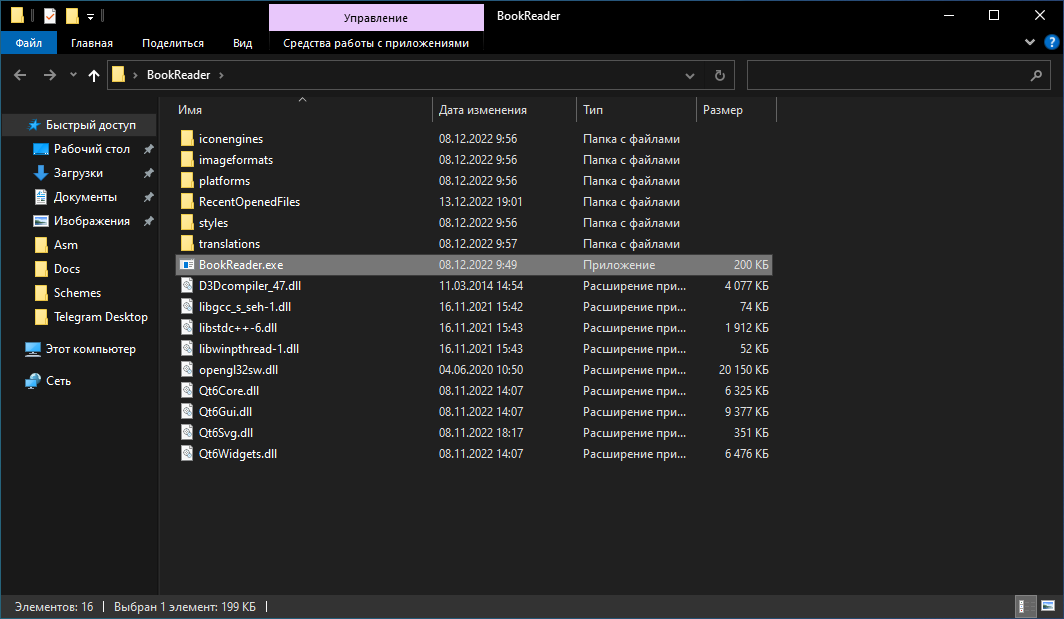


Рисунок 4.1 – Файл BookReader.exe

После запуска приложения появится главное окно (рис 4.2). Для того, чтобы начать читать, необходимо:

1. Нажать кнопку “Выбрать файл” в левой части окна;
2. В открывшемся диалоговом окне выбрать файл, имеющий расширения .fb2;
3. Нажать кнопку “Открыть”.

После открытия файла можно перейти на следующую страницы, нажав

кнопку -> (рисунок 4.2); перейти на предыдущую страницу, нажав кнопку <-; ввести в поле внизу экрана номер страницы, на которую необходимо перейти, и нажать на кнопку с изображенной на ней лупой.

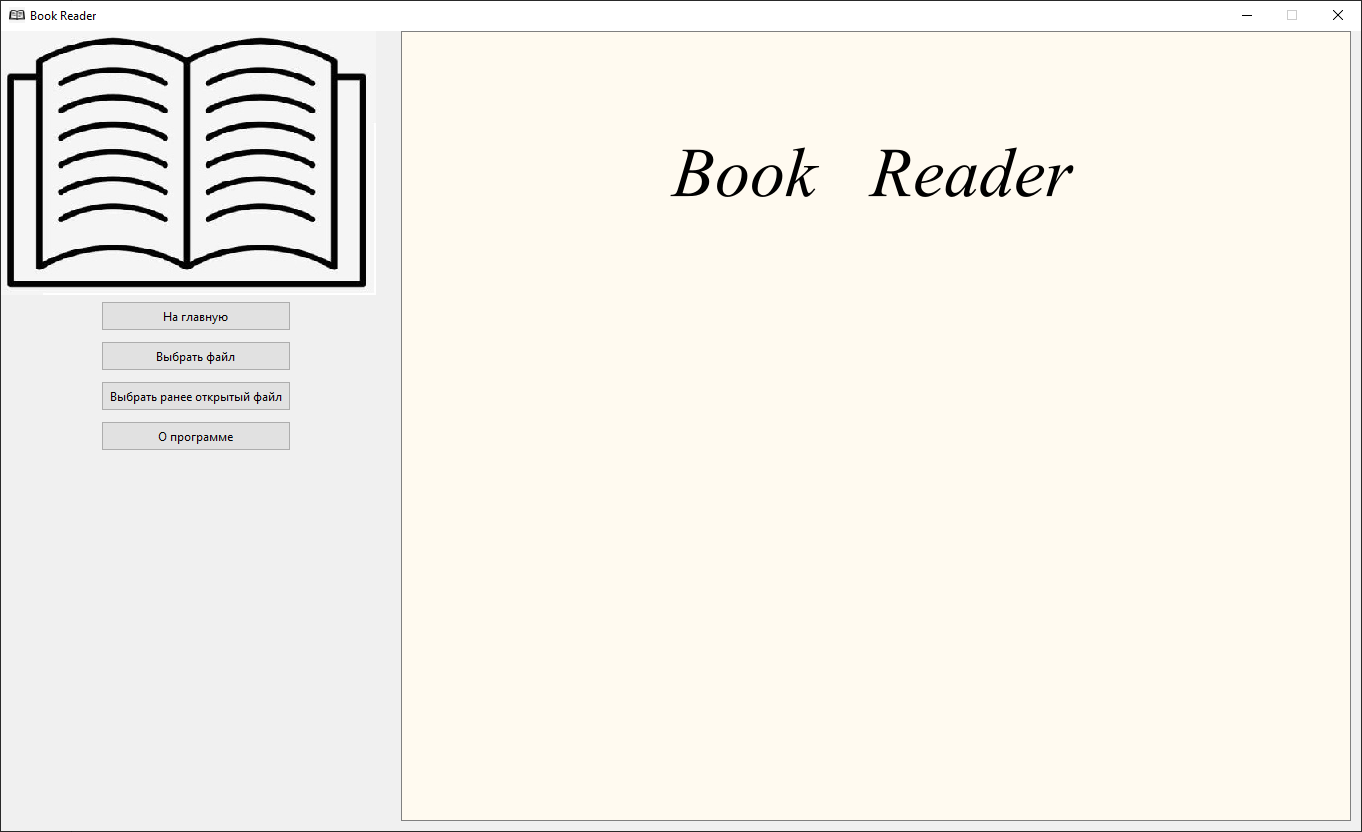


Рисунок 4.2 — Главное окно приложения.

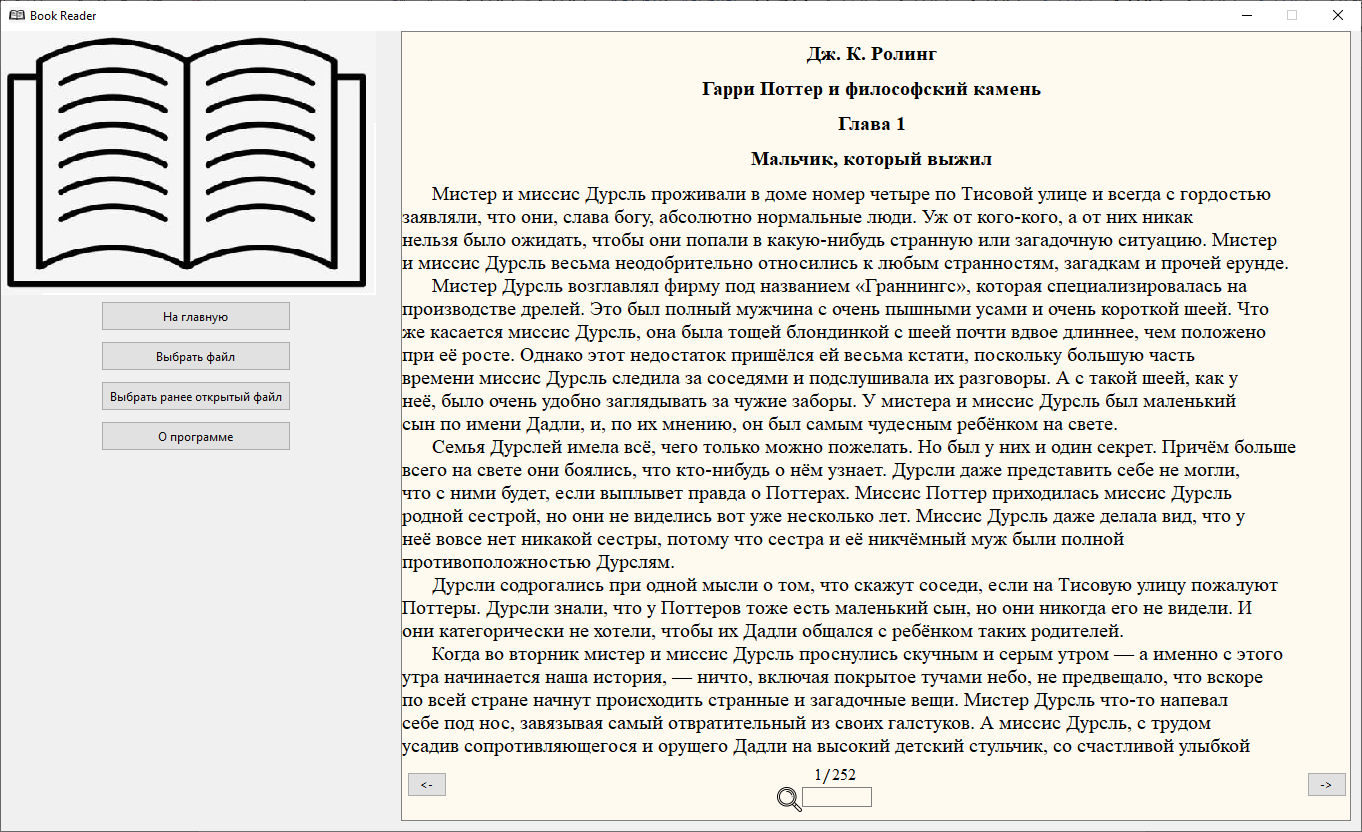


Рисунок 4.3 — Окно с открытой книгой.

Также в программе реализована функция открытия ранее открытых файлов в случае, если таковые имеются. Для этого необходимо нажать кнопку “Выбрать ранее открытый файл” и в появившемся диалоговом окне выбрать любой из ранее открытых файлов. После этого выбранный файл будет открыть на последней открытой странице. При необходимости ранее выбранные файлы можно удалять. Для этого необходимо нажать на кнопку с изображенной на ней урной.

Подробнее с функционалом приложения можно ознакомиться в приложении Б.

# ЗАКЛЮЧЕНИЕ

В ходе данного курсового проекта была написана программа для чтения электронных книг в формате fb2. Для написания программы были изучены фреймворк Qt, с помощью которого создавался пользовательский интерфейс приложения, и язык программирования C++, с помощью которого создавалась логика приложения.

Несмотря на то, что программа имеет довольно удобный пользовательский интерфейс, возможность открывать электронные книги в формате fb2, а также выбирать любую из ранее открытых книг на последней открытой странице, ее можно улучшить следующим образом:

- добавление возможности открыть электронные книги в различных форматах (epub, pdf);

- добавление возможности масштабировать текст;

- добавление возможности меня такие атрибуты текста, как шрифт и его размер;

- поиск текста по книге или по странице.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

[1] Страуструп, Б. Язык программирования С++/ Б.Страуструп; специальное издание. Пер. с англ. — СПб.: BHV, 2008. – 1098 с.

[2] Лафоре, Р. Объектно-ориентированное программирование в C++, 4-е издание / Р. Лафоре — СПб.: Питер, 2004.

[3] Официальный сайт документации Qt [Электронный ресурс]. —

Режим доступа - <https://doc.qt.io/.> – [Дата доступа] – 15.12.2022.

# ПРИЛОЖЕНИЕ А

(*обязательное*)

Листинг программы с комментариями

// Book.h:

#ifndef BOOK\_H

#define BOOK\_H

#include <QString>

#include <QXmlStreamReader>

#include "List.h"

// класс, содержащий в себе информацию о книге

class Book

{

public:

Book() = default;

~Book() = default;

void SetPathToBookFile(QString);

QString GetPathToBookFile() const;

void ParseBookFile();

List<QString> GetBookText() const;

void ResetBookText();

int GetTotalPagesNumber() const;

void SetCurrentPageNumber(int);

int GetCurrentPageNumber() const;

int GetMaxStringAmount() const;

Book& operator=(const Book&);

private:

void SetBookText(QXmlStreamReader&);

void SetTitleStyle(QXmlStreamReader&);

void SetParagraphStyle(QString);

void SetTotalPagesNumber();

QString pathToBookFile;

List<QString> bookText;

const int MAX\_STRING\_LENGTH = 95;

const int MAX\_STRING\_AMOUNT = 29;

int totalPagesNumber;

int currentPageNumber;

};

#endif

// Book.cpp:

#include <QFile>

#include <QMessageBox>

#include "../include/Book.h"

#include "../include/Exception.h"

void Book::SetPathToBookFile(QString \_pathToBookFile)

{

if (\_pathToBookFile.isEmpty())

{

throw ArgumentNullException("Файл не выбран!");

}

pathToBookFile = \_pathToBookFile;

}

QString Book::GetPathToBookFile() const

{

return pathToBookFile;

}

void Book::ParseBookFile()

{

QFile bookFile(pathToBookFile);

if (!bookFile.open(QIODevice::ReadOnly | QIODevice::Text)) // попытка открыть файл

{

QMessageBox::critical(nullptr, "", "Что-то пошло не так. Файл невозможно открыть!");

return;

}

QXmlStreamReader xmlFile(&bookFile);

while (!xmlFile.atEnd())

{

xmlFile.readNextStartElement();

if (xmlFile.isEndElement()) // перейти на следующую итерацию, если следующий считанный элемент является закрывающим тегом

{

continue;

}

QString tag = xmlFile.name().toString();

if (tag == "body")

{

SetBookText(xmlFile); // если найден тег "body" задать текст книги

break;

}

}

bookFile.close();

}

void Book::SetBookText(QXmlStreamReader& xmlFile)

{

QString tag;

while (tag != "body") // цикл до тех пор, пока не найден конец тега "body"

{

xmlFile.readNextStartElement(); // считать следующий тег

tag = xmlFile.name().toString(); // считанный тег

if (xmlFile.isEndElement()) // перейти на следующую итерацию, если следующий считанный элемент является закрывающим тегом

{

continue;

}

while (tag != "p" && tag != "title") // цикл до тех пор, пока не найден тег "P" (абзац) или тег "title" (заголовок)

{

xmlFile.readNextStartElement();

tag = xmlFile.name().toString();

}

if (tag == "p")

{ SetParagraphStyle(xmlFile.readElementText(QXmlStreamReader::ReadElementTextBehaviour::IncludeChildElements)); // если тег "p" - задать стиль текста абзаца

}

else

{

SetTitleStyle(xmlFile); // иначе задать стиль текста заголовка

}

}

SetTotalPagesNumber(); // установить общее число страниц книги

}

void Book::SetTotalPagesNumber()

{ // метод установки общего числа страниц книги

if (bookText.GetSize() % MAX\_STRING\_AMOUNT == 0)

{

totalPagesNumber = bookText.GetSize() / MAX\_STRING\_AMOUNT;

}

else

{

totalPagesNumber = bookText.GetSize() / MAX\_STRING\_AMOUNT + 1;

}

}

List<QString> Book::GetBookText() const

{

return bookText;

}

void Book::SetTitleStyle(QXmlStreamReader& xmlFile)

{

QString tag;

while (tag != "title") // цикл до тех пор, пока не найден конец тега "title"

{

xmlFile.readNextStartElement();

tag = xmlFile.name().toString();

if (xmlFile.isEndElement()) // перейти на следующую итерацию, если следующий считанный элемент является закрывающим тегом

{

continue;

}

while (tag != "p") // цикл до тех пор, пока не найден текста тега "title"

{

xmlFile.readNextStartElement();

tag = xmlFile.name().toString();

}

// задать элементу тега "title" жирный шрифт, выравненный по центру

bookText.Append("<p align = 'center'><b>" + xmlFile.readElementText(QXmlStreamReader::ReadElementTextBehaviour::IncludeChildElements) +

"</p></b>");

}

}

void Book::SetParagraphStyle(QString paragraphText)

{

if (paragraphText.length() <= MAX\_STRING\_LENGTH) // если абзац меньше, чем максимальная длина одной строки

{

bookText.Append("<p style = \"margin-left: 30; margin-right: 0; margin-top: 0; margin-bottom: 0\">" + paragraphText + "</p>");

return;

}

bool isFirstLineOfParagraph = true;

while (paragraphText.length() > MAX\_STRING\_LENGTH) // пока абзац длинне, чем максимальная длина одной строки

{

QString textToAppend; // текст строки абзаца

for (int i = 0; i < MAX\_STRING\_LENGTH; i++)

{

textToAppend.append(paragraphText[i]);

}

int length = textToAppend.length();

length--;

while (textToAppend[length] != ',' && textToAppend[length] != ';' && textToAppend[length] != ':' && textToAppend[length] != '.'

&& textToAppend[length] != '?' && textToAppend[length] != '!' && (textToAppend[length] != ' ' && paragraphText[length] != ' '))

{ // удалять символ строки до тех пор, пока он не равен знаку препинания, пробелу или концу слова

textToAppend.remove(length, 1);

length--;

}

length++;

paragraphText = paragraphText.remove(0, length); // удалить из текста абзаца считанную строку

if (isFirstLineOfParagraph)

{ // задать текст строки с отступом, если первая строка абзаца

bookText.Append("<p style = \"margin-left: 30; margin-right: 0; margin-top: 0; margin-bottom: 0\">" + textToAppend + "</p>");

isFirstLineOfParagraph = false;

}

else

{

bookText.Append("<p style = \"margin: 0\">" + textToAppend + "</p>"); // задать текст строки

}

}

bookText.Append("<p style = \"margin: 0\">" + paragraphText + "</p>"); // задать текст последней строки абзаца

}

void Book::ResetBookText()

{

bookText.Clear(); // удалить текст книги

}

int Book::GetTotalPagesNumber() const

{

return totalPagesNumber;

}

int Book::GetCurrentPageNumber() const

{

return currentPageNumber;

}

void Book::SetCurrentPageNumber(int \_currentPageNumber)

{

if (\_currentPageNumber > totalPagesNumber || \_currentPageNumber < 1)

{

throw OutOfRangeException("Вы не можете открыть данную страницу!");

}

currentPageNumber = \_currentPageNumber;

}

int Book::GetMaxStringAmount() const

{

return MAX\_STRING\_AMOUNT;

}

Book& Book::operator=(const Book& book)

{

this->currentPageNumber = book.currentPageNumber;

this->pathToBookFile = book.pathToBookFile;

return \*this;

}

// Exception.h:

#ifndef EXCEPTION\_H

#define EXCEPTION\_H

#include <QString>

class Exception

{

public:

Exception(QString);

Exception() = default;

QString GetErrorMessage() const;

private:

QString errorMessage;

};

class ArgumentNullException : public Exception

{

public:

ArgumentNullException(QString);

~ArgumentNullException() = default;

};

class OutOfRangeException : public Exception

{

public:

OutOfRangeException(QString);

~OutOfRangeException() = default;

};

#endif

// Exception.cpp:

#include "../include/Exception.h"

Exception::Exception(QString \_errorMessage) : errorMessage(\_errorMessage)

{ }

QString Exception::GetErrorMessage() const

{

return errorMessage;

}

ArgumentNullException::ArgumentNullException(QString \_errorMessage) : Exception(\_errorMessage)

{ }

OutOfRangeException::OutOfRangeException(QString \_errorMessage) : Exception(\_errorMessage)

{ }

// List.h:

#ifndef LIST\_H

#define LIST\_H

template<class T>

class Iterator;

template<class T>

class Node

{

public:

T data;

Node<T>\* next;

Node (T);

~Node() = default;

};

template<class T>

Node<T>::Node(T \_data) : data(\_data)

{ }

template<class T>

class List

{

public:

List();

~List() = default;

void Append(T);

void Pop();

void Clear();

T operator[](int);

int GetSize() const;

friend class Iterator<T>;

private:

Node<T>\* head;

Node<T>\* tail;

int size;

};

template<class T>

List<T>::List() : head(nullptr), tail(nullptr), size(0)

{ }

template<class T>

void List<T>::Append(T data)

{

Node<T>\* temp = new Node(data);

if (!head)

{

head = temp;

tail = temp;

}

else

{

tail->next = temp;

tail = temp;

}

size++;

}

template<class T>

void List<T>::Pop()

{

if (head)

{

Node<T>\* temp = head;

head = head->next;

delete temp;

}

size--;

}

template<class T>

void List<T>::Clear()

{

int tmpSize = size;

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

{

Pop();

}

head = nullptr;

tail = nullptr;

}

template<class T>

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

{

Iterator<T> iterator = head;

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

{

++iterator;

}

return \*iterator;

}

template<class T>

int List<T>::GetSize() const

{

return size;

}

template<class T>

class Iterator

{

public:

Iterator(Node<T>\*);

~Iterator() = default;

Iterator<T>& operator++();

T& operator\*() const;

private:

Node<T>\* currentNode;

};

template<class T>

Iterator<T>::Iterator(Node<T>\* node) : currentNode(node)

{ }

template<class T>

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

{

currentNode = currentNode->next;

return \*this;

}

template<class T>

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

{

return currentNode->data;

}

#endif

// MainWindow.h:

#ifndef MAIN\_WINDOW\_H

#define MAIN\_WINDOW\_H

#include <QMainWindow>

#include "Book.h"

namespace Ui { class MainWindow; }

// класс для работы с главным окном

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget\* parent = nullptr);

~MainWindow();

private slots:

void on\_aboutProgramButton\_clicked();

void on\_toStartMenuButton\_clicked();

void on\_chooseFileButton\_clicked();

void on\_nextPageButton\_clicked();

void on\_previousPageButton\_clicked();

void on\_findPageButton\_clicked();

void on\_chooseRecentOpenedFileButton\_clicked();

private:

void ConfigureMainWindow();

void ConfigureTabWidget();

void ConfigureMainWindowLabel();

void ConfigureAboutProgramLabel();

void ConfigureBookTab();

void ConfigurePageButtons();

void SetBookLabelText(List<QString>, int);

void AppendPageNumberToBookFile(int);

Ui::MainWindow\* ui;

Book book;

const int WINDOW\_WIDTH = 1360;

const int WINDOW\_HEIGHT = 800;

};

#endif

// MainWindow.cpp:

#include <QFileDialog>

#include <QMessageBox>

#include <QFile>

#include "../include/MainWindow.h"

#include "../include/RecentOpenedFilesWindow.h"

#include "../include/WidgetStyle.h"

#include "../include/Exception.h"

#include "ui\_mainwindow.h"

MainWindow::MainWindow(QWidget\* parent) : QMainWindow(parent), ui(new Ui::MainWindow)

{

ui->setupUi(this);

ConfigureMainWindow();

ConfigureTabWidget();

ConfigureMainWindowLabel();

ConfigureAboutProgramLabel();

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_aboutProgramButton\_clicked()

{

ui->programTab->setCurrentIndex(2); // открыть меню с информацией

}

void MainWindow::on\_toStartMenuButton\_clicked()

{

ui->programTab->setCurrentIndex(0); // открыть стартовое меню

}

void MainWindow::on\_chooseFileButton\_clicked()

{

QString path = QFileDialog::getOpenFileName(this, "Выберите файл", "C:", "Book file (\*.fb2)"); // открыть диалоговое окно для выбора fb2 файла

try

{

book.SetPathToBookFile(path); // попытка задать путь до файла

}

catch (const ArgumentNullException& exception)

{

QMessageBox::warning(this, "", exception.GetErrorMessage());

return;

}

book.ResetBookText(); // обнулить текст книги

book.ParseBookFile(); // считать текст книги

book.SetCurrentPageNumber(1); // задать номер текущей страницы

ConfigureBookTab(); // настроить стили меню с книгой

SetBookLabelText(book.GetBookText(), 1); // отобразить текст страницы на окне

QString bookFileName = (QFileInfo(QFile(path).fileName())).fileName();

if (!QFile::exists("RecentOpenedFiles/" + bookFileName))

{

if (!QDir("RecentOpenedFiles").exists())

{

QDir().mkpath("RecentOpenedFiles"); // если директория для ранее открытых файлов не создана, создать ее

}

QFile::copy(path, "RecentOpenedFiles/" + bookFileName); // создать копию файла для возможности открыть его на последней открытой странице

}

AppendPageNumberToBookFile(book.GetCurrentPageNumber()); // добавить номер страницы в конец файла книги

}

void MainWindow::on\_nextPageButton\_clicked()

{

int currentPageNumber = book.GetCurrentPageNumber() + 1;

try

{

book.SetCurrentPageNumber(currentPageNumber); // попытка задать новый номер текущей страницы

}

catch (const OutOfRangeException& exception)

{

QMessageBox::warning(this, "", exception.GetErrorMessage());

return;

}

ui->currentPageNumberLabel->setText(QString::number(currentPageNumber)); // отобразить номер текущей страницы

SetBookLabelText(book.GetBookText(), currentPageNumber); // отобразить текст новой страницы

AppendPageNumberToBookFile(book.GetCurrentPageNumber()); // добавить номер страницы в конец файла книги

}

void MainWindow::on\_previousPageButton\_clicked()

{

int currentPageNumber = book.GetCurrentPageNumber() - 1;

try

{

book.SetCurrentPageNumber(currentPageNumber); // попытка задать новый номер текущей страницы

}

catch (const OutOfRangeException& exception)

{

QMessageBox::warning(this, "", exception.GetErrorMessage());

return;

}

ui->currentPageNumberLabel->setText(QString::number(currentPageNumber)); // отобразить номер текущей страницы

SetBookLabelText(book.GetBookText(), currentPageNumber); // отобразить текст новой страницы

AppendPageNumberToBookFile(book.GetCurrentPageNumber()); // добавить номер страницы в конец файла книги

}

void MainWindow::on\_findPageButton\_clicked()

{

int pageNumber = ui->findPageLineEdit->text().toInt();

ui->findPageLineEdit->setText(""); // обнулить текст после ввода

try

{

book.SetCurrentPageNumber(pageNumber);

}

catch (const OutOfRangeException& exception)

{

QMessageBox::warning(this, "", exception.GetErrorMessage());

return;

}

ui->currentPageNumberLabel->setText(QString::number(pageNumber)); // отобразить номер текущей страницы

SetBookLabelText(book.GetBookText(), pageNumber); // отобразить текст новой страницы

AppendPageNumberToBookFile(pageNumber); // добавить номер страницы в конец файла книги

}

void MainWindow::on\_chooseRecentOpenedFileButton\_clicked()

{

ui->programTab->setCurrentIndex(0);

QDir recentOpenedFilesDirectory("RecentOpenedFiles/");

if (recentOpenedFilesDirectory.isEmpty())

{

QMessageBox::warning(this, "", "Ранее открытые файлы отсутсвуют!"); // если ранее открытые файлы отсутсвуют

return;

}

RecentOpenedFilesWindow window(this);

window.show();

window.exec();

try

{

book = window.GetBook(); // попытка получить книгу, если она была выбрана

}

catch (const ArgumentNullException& exception)

{

QMessageBox::warning(this, "", exception.GetErrorMessage());

return;

}

book.ResetBookText(); // обнулить текст книги

book.ParseBookFile(); // считать текст книги

ConfigureBookTab(); // настроить стили меню с книгой

SetBookLabelText(book.GetBookText(), book.GetCurrentPageNumber()); // отобразить текст новой страницы

}

void MainWindow::ConfigureMainWindow()

{

setFixedSize(WINDOW\_WIDTH, WINDOW\_HEIGHT); // задать размер окна

setWindowTitle("Book Reader"); // задать заголовок окна

setWindowIcon(QIcon(":/img/program\_icon.png")); // задать икноку окна

QPixmap labelPicture(":/img/book\_image.jpg"); // отобразить картинку на окне

labelPicture = labelPicture.scaled(labelPicture.width() \* 0.8f, labelPicture.height() \* 0.8f);

ui->imageLabel->setFixedSize(labelPicture.size());

ui->imageLabel->setPixmap(labelPicture);

ui->toStartMenuButton->setStyleSheet(WidgetStyle::GetMainWindowButtonsStyle());

ui->chooseFileButton->setStyleSheet(WidgetStyle::GetMainWindowButtonsStyle());

ui->chooseRecentOpenedFileButton->setStyleSheet(WidgetStyle::GetMainWindowButtonsStyle());

ui->aboutProgramButton->setStyleSheet(WidgetStyle::GetMainWindowButtonsStyle());

}

void MainWindow::ConfigureTabWidget()

{

ui->programTab->setStyleSheet(WidgetStyle::GetTabStyle()); // задать стили для QTabWidget

ui->programTab->setCurrentIndex(0);

}

void MainWindow::ConfigureMainWindowLabel()

{

ui->mainWindowLabel->setStyleSheet(WidgetStyle::GetMainWindowLabelStyle()); // задать стили для стартового окна

ui->mainWindowLabel->setText("Book Reader"); // задать текст на стартовом окне

}

void MainWindow::ConfigureAboutProgramLabel()

{

ui->aboutProgramLabel->setStyleSheet(WidgetStyle::GetAboutPogramLabelStyle()); // задать стили для меню с информацией

ui->aboutProgramLabel->setText(" Эта программа предназначена для чтения электронных книг, имеющих расширение .fb2." // устноавить текст информации

" Для того, чтобы начать читать, необходимо:\n"

"1) нажать на кнопку \"Выбрать файл\" в левой части окна;\n"

"2) в открывшемся диалоговом окне выбрать файл, имеющий расширение .fb2;\n"

"3) выбрав нужный файл, нажать кнопку \"Открыть\".\n"

"После открытия файла для перехода на следующую страницу необходимо нажать на кнопку \"->\" в нижней части экрана, "

"для перехода на предыдущую страницу — нажать на кнопку \"<-\".\n"

" Если необходимо перейти на конкретную страницу, то необходимо в поле снизу ввести номер страницы и "

"нажать на кнопку с изображенной на ней лупой.\n"

" При нажатии на кнопку \"Выбрать ранее открытый файл\" можно открыть один из ранее открытых файлов на"

"последней открытой странице либо удалить этот файл.");

}

void MainWindow::ConfigureBookTab()

{ // задать стили для меню с книгой

ui->programTab->setCurrentIndex(1);

ui->bookLabel->setStyleSheet(WidgetStyle::GetBookLabelStyle());

// задать стили для части окна с номерами текущего и обшего числа страниц

ui->totalPagesNumberLabel->setStyleSheet(WidgetStyle::GetPageNumberLabelsStyle());

ui->totalPagesNumberLabel->setText(QString::number(book.GetTotalPagesNumber()));

ui->currentPageNumberLabel->setStyleSheet(WidgetStyle::GetPageNumberLabelsStyle());

ui->currentPageNumberLabel->setText(QString::number(book.GetCurrentPageNumber()));

ui->slashLabel->setStyleSheet("font-size: 18px;");

// задать стили для меню поиска страницы

QIcon findPageButtonIcon(":/img/magnifier\_icon.png");

ui->findPageButton->setIcon(findPageButtonIcon);

ui->findPageButton->setIconSize(ui->findPageButton->size());

ui->findPageButton->setStyleSheet(WidgetStyle::GetFindPageButtonStyle());

ui->findPageLineEdit->setStyleSheet(WidgetStyle::GetFindPageLineEditStyle());

}

void MainWindow::SetBookLabelText(List<QString> bookText, int currentPageNumber)

{

QString pageText;

int maxStringAmount = book.GetMaxStringAmount();

for (int i = 0; i < maxStringAmount; i++) // пока не будет добавлено максимальное число строк к тексту страницы

{

int index = (currentPageNumber - 1) \* maxStringAmount + i;

if (index > bookText.GetSize() - 1) // если на последней странице строк меньше, чем из максимальное количество

{

break;

}

pageText.append(bookText[index]); // добавить строку к тексту страницы

}

ui->bookLabel->setText(pageText); // отобразить текст страницы

}

void MainWindow::AppendPageNumberToBookFile(int pageNumber)

{

QString bookFileName = (QFileInfo(QFile(book.GetPathToBookFile()).fileName())).fileName(); // получить имя файла

QFile bookFile("RecentOpenedFiles/" + bookFileName);

bookFile.open(QIODevice::Text | QIODevice::Append); // открыть файл

QTextStream bookFileStream(&bookFile);

bookFileStream << '\n' << pageNumber; // добавить номер страницы в файл

}

// RecentOpenedFilesWindow.h:

#ifndef RECENT\_OPENED\_FILES\_WINDOW\_H

#define RECENT\_OPENED\_FILES\_WINDOW\_H

#include <QDialog>

#include <QPushButton>

#include "Book.h"

namespace Ui { class RecentOpenedFilesWindow; }

// класс для работы с окном со списком ранее открытых файлов

class RecentOpenedFilesWindow : public QDialog

{

Q\_OBJECT

public:

RecentOpenedFilesWindow(QWidget \*parent = nullptr);

~RecentOpenedFilesWindow();

Book GetBook() const;

private slots:

void RecentOpenedFileButtonClicked();

void DeleteRecentOpenedFileButtonClicked();

private:

void ConfigureDialogWindow();

void SetRecentOpenedFiles();

void SetLastOpenedPageNumber(QString);

void CreateRecentOpenedFilesButtons();

void CreateDeleteRecentOpenedFilesButtons();

void ChangeRecentOpenedFilesButtonsPosition(int);

Ui::RecentOpenedFilesWindow \*ui;

const int WINDOW\_WIDTH = 800;

const int WINDOW\_HEIGHT = 600;

QList<QString> recentOpenedFiles;

QList<QPushButton\*> recentOpenedFilesButtons;

QList<QPushButton\*> deleteRecentOpenedFilesButtons;

Book book;

};

#endif

// RecentOpenedFilesWindow.cpp:

#include <QDir>

#include "../include/RecentOpenedFilesWindow.h"

#include "../include/WidgetStyle.h"

#include "../include/Exception.h"

#include "ui\_recentopenedfileswindow.h"

RecentOpenedFilesWindow::RecentOpenedFilesWindow(QWidget\* parent) : QDialog(parent), ui(new Ui::RecentOpenedFilesWindow)

{

ui->setupUi(this);

ConfigureDialogWindow();

SetRecentOpenedFiles();

CreateRecentOpenedFilesButtons();

CreateDeleteRecentOpenedFilesButtons();

}

RecentOpenedFilesWindow::~RecentOpenedFilesWindow()

{

delete ui;

}

Book RecentOpenedFilesWindow::GetBook() const

{

if ((book.GetPathToBookFile()).isEmpty())

{

throw ArgumentNullException("Файл не выбран");

}

return book;

}

void RecentOpenedFilesWindow::RecentOpenedFileButtonClicked()

{

QPushButton\* btn = (QPushButton\*)sender();

book.SetPathToBookFile("RecentOpenedFiles/" + (btn->text()).append(".fb2")); // получить имя выбранной книги

book.ParseBookFile();

SetLastOpenedPageNumber((btn->text()).append(".fb2")); // задать последнюю страницы выбранной книги

}

void RecentOpenedFilesWindow::DeleteRecentOpenedFileButtonClicked()

{

QPushButton\* btn = (QPushButton\*)sender();

int deleteButtonY = btn->y(); // получить y координату выбранной кнопки удалить (она равна y координате кнопки с книгой)

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

{

if (deleteButtonY == recentOpenedFilesButtons[i]->y())

{

ChangeRecentOpenedFilesButtonsPosition(i); // если найден номер кнопки с названием книги, который необходимо удалить

break;

}

}

}

void RecentOpenedFilesWindow::ConfigureDialogWindow()

{

setFixedSize(WINDOW\_WIDTH, WINDOW\_HEIGHT);

setWindowTitle("Ранее открытые файлы");

setWindowIcon(QIcon(":/img/program\_icon.png"));

setWindowModality(Qt::ApplicationModal);

}

void RecentOpenedFilesWindow::SetRecentOpenedFiles()

{

QDir recentOpenedFilesDirectory("RecentOpenedFiles/");

recentOpenedFiles = recentOpenedFilesDirectory.entryList(QDir::NoDotAndDotDot | QDir::AllEntries); // получить список всех ранее открытых файлов из директории

}

void RecentOpenedFilesWindow::CreateRecentOpenedFilesButtons()

{

const int BUTTON\_WIDTH = WINDOW\_WIDTH; // длина кнопки с названием книги

const int BUTTON\_HEIGHT = 30; // высота кнопки с названием книги

for (int i = 0; i < recentOpenedFiles.size(); i++) // цикл по всем ранее открытым книгам

{

recentOpenedFilesButtons.append(new QPushButton(this)); // добавление кнопки в список

recentOpenedFilesButtons[i]->setText(recentOpenedFiles[i].remove(recentOpenedFiles[i].size() - 4, 4)); // текст кнопки - имя файла без расширения

recentOpenedFilesButtons[i]->setFixedSize(BUTTON\_WIDTH, BUTTON\_HEIGHT); // установить размеры кнопки

recentOpenedFilesButtons[i]->move(0, i \* BUTTON\_HEIGHT); // задать координаты кнопки

recentOpenedFilesButtons[i]->setStyleSheet(WidgetStyle::GetRecentOpenedFilesButtonsStyle()); // установка стилей для кнопки

// обработка нажатия кнопки

connect(recentOpenedFilesButtons[i], SIGNAL(clicked()), this, SLOT(RecentOpenedFileButtonClicked()));

connect(recentOpenedFilesButtons[i], SIGNAL(clicked()), this, SLOT(close()));

}

}

void RecentOpenedFilesWindow::CreateDeleteRecentOpenedFilesButtons()

{

const int BUTTON\_WIDTH = 50; // длина кнопки удаления

const int BUTTON\_HEIGHT = 30; // высота кнопки удаления

for (int i = 0; i < recentOpenedFiles.size(); i++) // цикл по всем ранее открытым книгам

{

deleteRecentOpenedFilesButtons.append(new QPushButton(this)); // добавление кнопки в список

deleteRecentOpenedFilesButtons[i]->setFixedSize(BUTTON\_WIDTH, BUTTON\_HEIGHT); // установить размеры кнопки

deleteRecentOpenedFilesButtons[i]->move(WINDOW\_WIDTH - 50, i \* BUTTON\_HEIGHT); // задать координаты кнопки

deleteRecentOpenedFilesButtons[i]->setStyleSheet(WidgetStyle::GetDeleteRecentOpenedFilesButtonsStyle()); // установка стилей для кнопки

// создание иконки кнопки

QIcon deleteRecentOpenedFileButtonIcon(":/img/delete\_file\_icon.png");

deleteRecentOpenedFilesButtons[i]->setIcon(deleteRecentOpenedFileButtonIcon);

deleteRecentOpenedFilesButtons[i]->setIconSize(deleteRecentOpenedFilesButtons[i]->size());

// обработка нажатия кнопки

connect(deleteRecentOpenedFilesButtons[i], SIGNAL(clicked()), this, SLOT(DeleteRecentOpenedFileButtonClicked()));

}

}

void RecentOpenedFilesWindow::SetLastOpenedPageNumber(QString fileName)

{

QFile bookFile("RecentOpenedFiles/" + fileName);

bookFile.open(QIODevice::Text | QIODevice::ReadOnly); // открытие выбранного файла

QTextStream bookFileStream(&bookFile);

QString bookFileText = bookFileStream.readAll(); // получить весь текст файла

QString lastOpenedPage;

for (int i = bookFileText.length() - 1; bookFileText[i] != '\n'; i--) // цикл до конца последней строки файла

{

lastOpenedPage.append(bookFileText[i]); // добавить символ с конца файла (в конец файла добавлялись номера страниц)

}

std::reverse(lastOpenedPage.begin(), lastOpenedPage.end()); // реверс строки

book.SetCurrentPageNumber(lastOpenedPage.toInt()); // перевод номера последней открытой страницы в int

}

void RecentOpenedFilesWindow::ChangeRecentOpenedFilesButtonsPosition(int i)

{

QFile::remove("RecentOpenedFiles/" + (recentOpenedFilesButtons[i]->text()).append(".fb2")); // удаление выбранного файла из директории

// закрытие кнопок, соответствующих выбранному файлу

recentOpenedFilesButtons[i]->close();

deleteRecentOpenedFilesButtons[i]->close();

// смещение кнопок, если нажатая кнопка не последняя

if (i < recentOpenedFilesButtons.size() - 1)

{

for (int j = i + 1; j < recentOpenedFilesButtons.size(); j++)

{

recentOpenedFilesButtons[j]->move(0, 30 \* (j - 1)); // сместить вверх все кнопки ниже той, которая была удалена

deleteRecentOpenedFilesButtons[j]->move(WINDOW\_WIDTH - 50, 30 \* (j - 1));

}

}

// удаление выбранного файла и кнопок из списков

recentOpenedFiles.removeAt(i);

recentOpenedFilesButtons.removeAt(i);

deleteRecentOpenedFilesButtons.removeAt(i);

}

// WidgetStyle.h:

#ifndef WIDGET\_STYLE\_H

#define WIDGET\_STYLE\_H

#include <QString>

// класс, содержащий в себе стили для всех виджетов

class WidgetStyle

{

public:

static QString GetTabStyle();

static QString GetAboutPogramLabelStyle();

static QString GetMainWindowLabelStyle();

static QString GetPageNumberLabelsStyle();

static QString GetBookLabelStyle();

static QString GetFindPageButtonStyle();

static QString GetFindPageLineEditStyle();

static QString GetRecentOpenedFilesButtonsStyle();

static QString GetDeleteRecentOpenedFilesButtonsStyle();

static QString GetMainWindowButtonsStyle();

};

#endif

// WidgetStyle.cpp:

#include "../include/WidgetStyle.h"

QString WidgetStyle::GetTabStyle()

{

return "QTabBar::tab {"

"width: 0;"

"height: 0;"

"}"

"QTabWidget::pane {"

"background: #FFFAF0;"

"border: 1px solid gray;"

"}";

}

QString WidgetStyle::GetAboutPogramLabelStyle()

{

return "QLabel {"

"qproperty-wordWrap: true;"

"font-size: 20px;"

"font-family: \"Times New Roman\";"

"}";

}

QString WidgetStyle::GetMainWindowLabelStyle()

{

return "QLabel {"

"qproperty-wordWrap: false;"

"font-size: 70px;"

"font-family: \"Times New Roman\";"

"font-style: italic;"

"font-weight: 500;"

"}";

}

QString WidgetStyle::GetPageNumberLabelsStyle()

{

return "QLabel {"

"qproperty-wordWrap: false;"

"font-size: 16px;"

"font-family: \"Times New Roman\";"

"}";

}

QString WidgetStyle::GetBookLabelStyle()

{

return "QLabel {"

"qproperty-wordWrap: true;"

"font-size: 20px;"

"font-family: \"Times New Roman\";"

"}";

}

QString WidgetStyle::GetFindPageButtonStyle()

{

return "QPushButton {"

"background: #FFFAF0;"

"border: 0px solid white;"

"}";

}

QString WidgetStyle::GetFindPageLineEditStyle()

{

return "QLineEdit {"

"background: #FFFAF0;"

"border: 1px solid gray;"

"font-size: 16px;"

"font-family: \"Times New Roman\";"

"}";

}

QString WidgetStyle::GetRecentOpenedFilesButtonsStyle()

{

return "QPushButton {"

"border: 1px solid gray;"

"font-size: 16px;"

"font-family: \"Times New Roman\";"

"}"

"QPushButton:hover {"

"border: 1px solid #1E90FF;"

"}"

"QPushButton::pressed {"

"border: 1px solid #005499;"

"background-color: #CCE4F7;"

"}";

}

QString WidgetStyle::GetDeleteRecentOpenedFilesButtonsStyle()

{

return "QPushButton {"

"border: 0px solid white;"

"}";

}

QString WidgetStyle::GetMainWindowButtonsStyle()

{

return "QPushButton {"

"font-size: 12px;"

"}";

}

// main.cpp:

#include <QApplication>

#include "../include/MainWindow.h"

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

{

QApplication app(argc, argv);

MainWindow w;

w.show();

return app.exec();

}

# ПРИЛОЖЕНИЕ Б

(*обязательное*)

Скриншоты работы программы

При запуске программы пользователь попадает на главное окно (рис. Б.1).

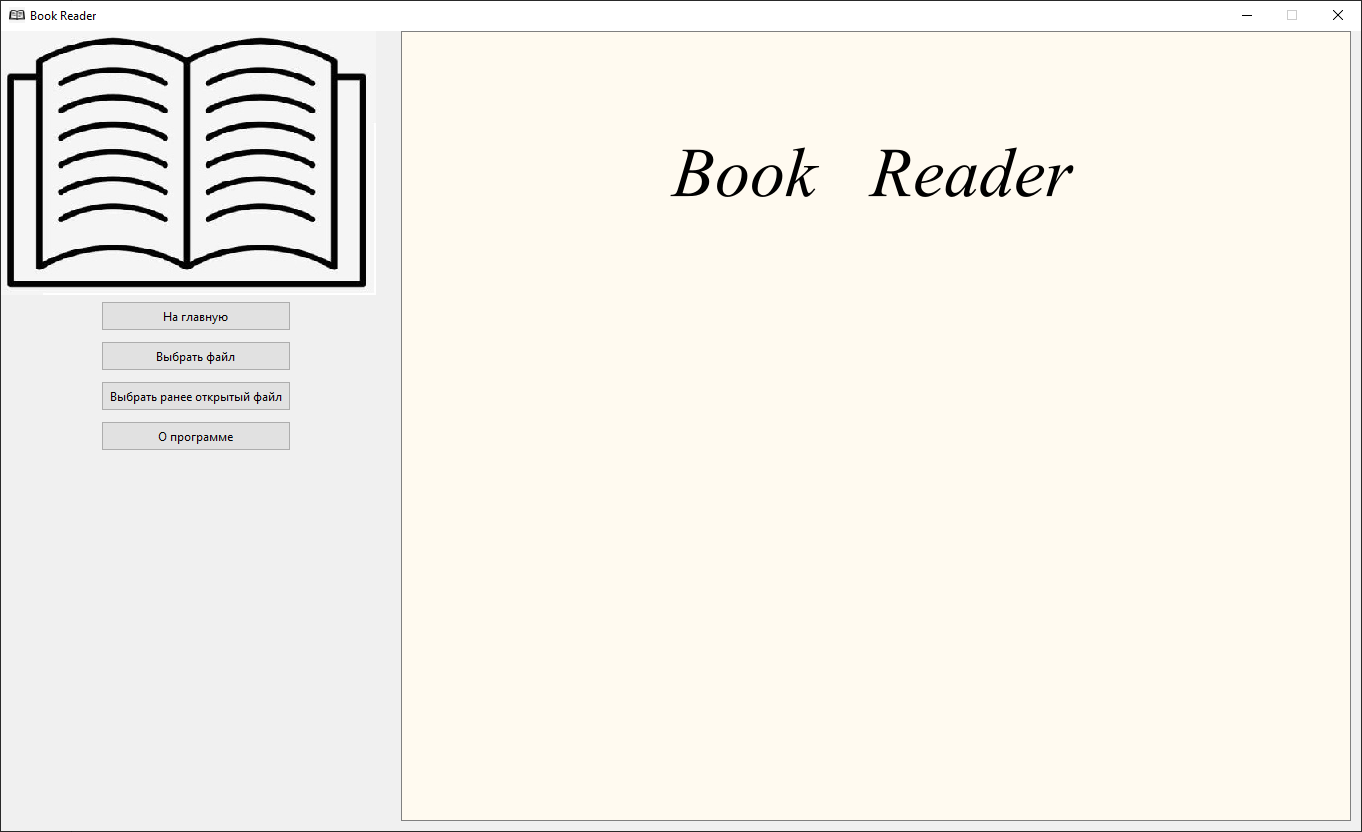


Рисунок Б.1 — Главное окно приложения.

При нажатии на кнопку “Выбрать файл” появляется меню с выбором файла (рис. Б.2). В данном меню есть фильтр, который не позволяет выбрать никакой файл, кроме .fb2.

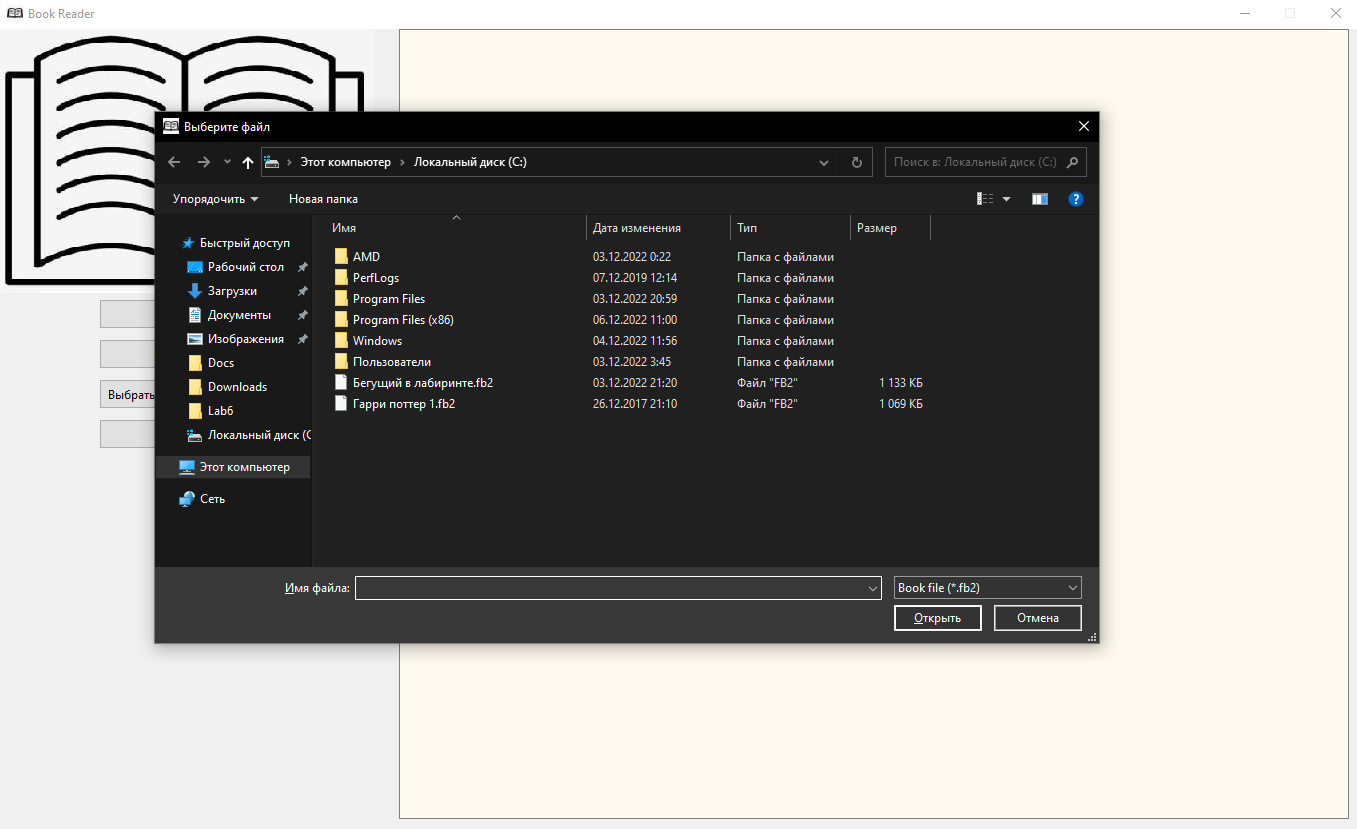


Рисунок Б.2 — Меню выбора файла.

Если данное окно было закрыто без выбора файла, появляется окно с предупреждением о том, что файл не выбран (рис. Б.3).

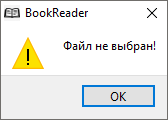


Рисунок Б.3 — Предупреждение об отсутствии выбранного файла.

При выборе файла на окне отображается текст файла (рис. Б.4).

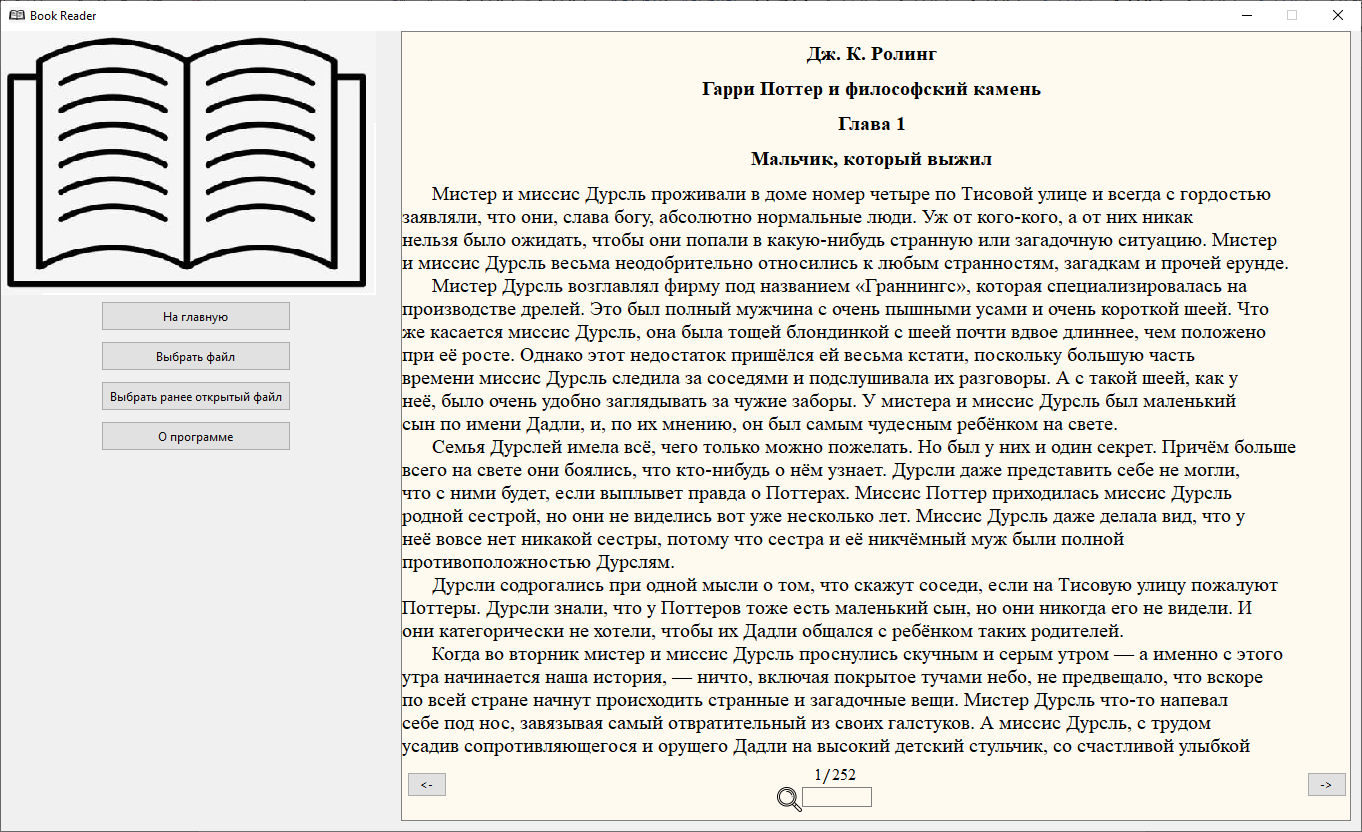


Рисунок Б.4 — Отображение текста выбранной книги.

При попытке перейти на предыдущую страницу, находясь на первой, перейти на следующую страницу, находясь на последней, или при неверном вводе страницы с клавиатуры появляется окно с предупреждением (рис. Б.5).

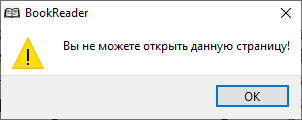


Рисунок Б.5 — Предупреждение при попытке перейти на несуществующую страницу.

При нажатии на кнопку “Выбрать ранее открытый файл” при их отсутствии появляется предупреждение (рис. Б.6).

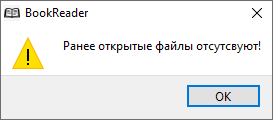


Рисунок Б.6 — Предупреждение при попытке выбрать ранее открытые файлы при их отсутствии.

Если ранее открытые файлы есть, появляется диалоговое окно, в котором можно выбрать один из файлов (рис. Б.7).

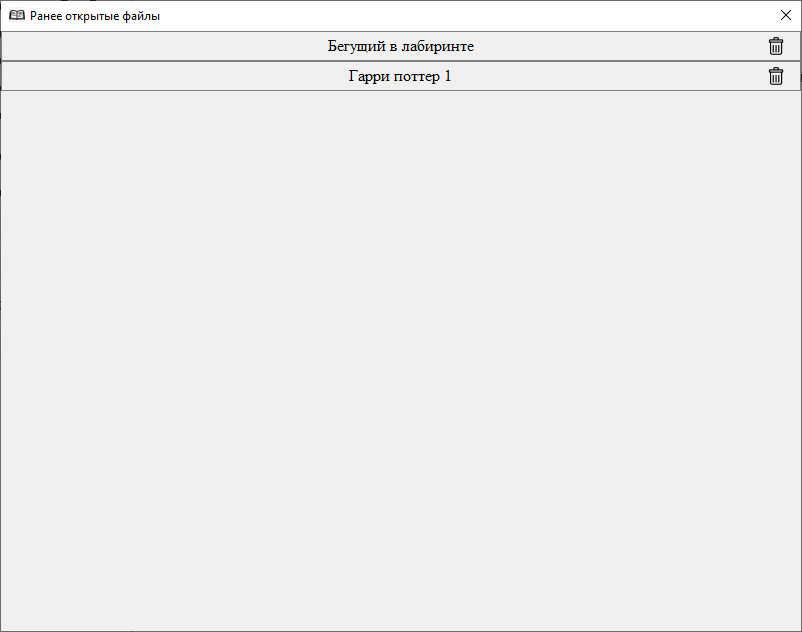


Рисунок Б.7 — Диалоговое окно для выбора ранее открытых файлов.

При выборе файла диалоговое окно закроется, и на главном окне появится текст выбранной книги на последней открытой странице (рис. Б.8).

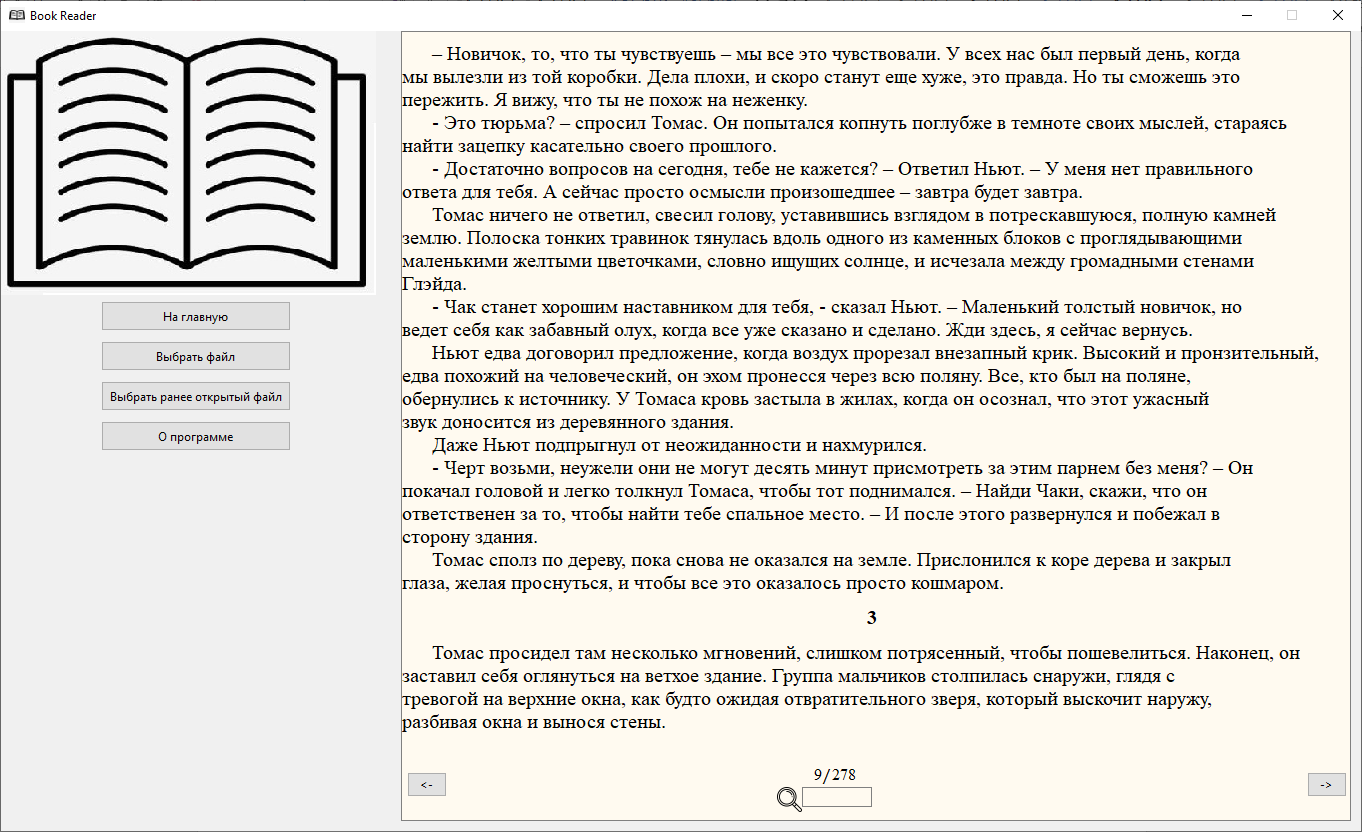


Рисунок Б.8 — Текст книги из меню ранее открытых файлов на последней открытой странице.

При необходимости можно удалить книгу из меню ранее открытых файлов (рис. Б.9).

Если данное окно было закрыто без выбора файла, появляется окно с предупреждением о том, что файл не выбран (рис. Б.3).

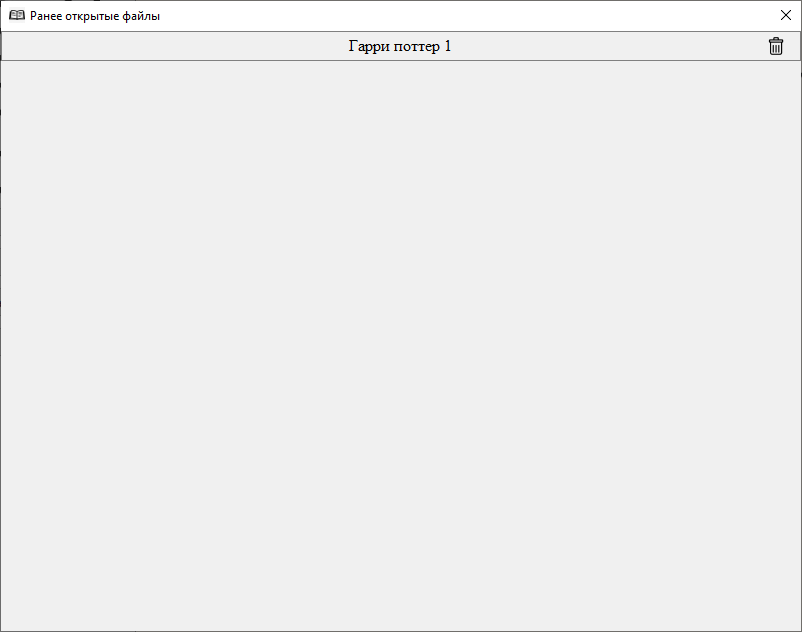


Рисунок Б.9 — Меню выбора ранее открытых файлов после удаления одного из файлов.

При нажатии на кнопку “О программе” появляется меню, в котором указан весь функционал приложения (рис. Б.10).

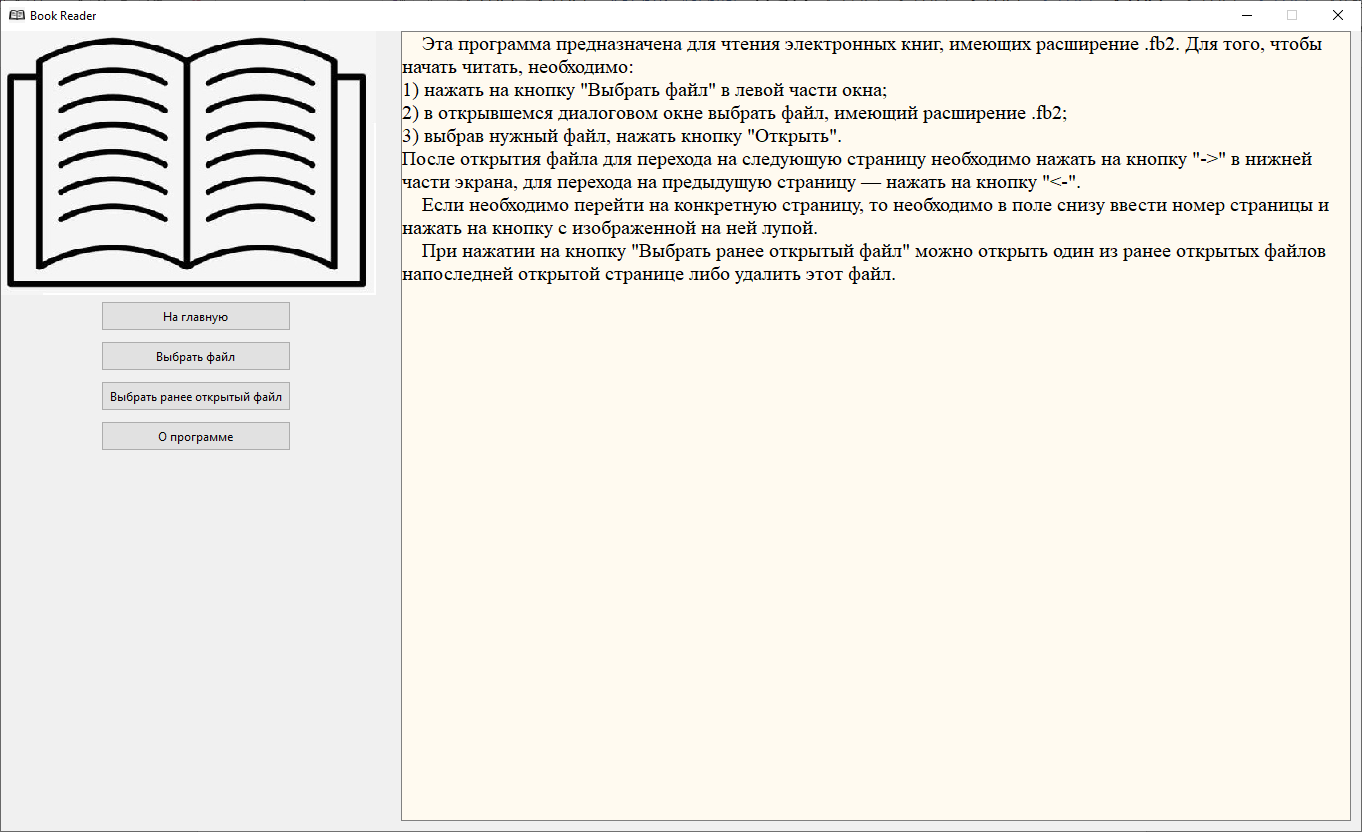


Рисунок Б.10 — Меню “О программе”.

ПРИЛОЖЕНИЕ В  
(о*бязательное*)  
  
Диаграмма классов

ПРИЛОЖЕНИЕ Г  
(о*бязательное*)  
  
Блок-схема алгоритмов

# ПРИЛОЖЕНИЕ Д

(о*бязательное*)

Ведомость документов