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

АРХИВАТОР

БГУИР КП 1-40 02 01 112 ПЗ

Студент: группы 150541,

Кипень В. В.

Руководитель: Ассистент кафедры ЭВМ,

Марзалюк А. В.

МИНСК 2023

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc153873850)

[1 ПОСТАНОВКА ЗАДАЧИ 6](#_Toc153873851)

[2 ОБЗОР ЛИТЕРАТУРЫ 7](#_Toc153873852)

[2.1 Обзор методов и алгоритмов поставленной задачи 8](#_Toc153873853)

[3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 11](#_Toc153873854)

[3.1. Структура входных и выходных данных 11](#_Toc153873855)

[3.2. Разработка диаграммы классов 11](#_Toc153873856)

[3.3. Описание классов 11](#_Toc153873857)

[3.3.1 Класс ClassAddToZIP 11](#_Toc153873858)

[3.3.2 Класс ClassExtractZIP 11](#_Toc153873859)

[3.3.3 Класс MainWindow 12](#_Toc153873860)

[4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 13](#_Toc153873861)

[4.1 Разработка схем алгоритмов (два наиболее важных метода) 13](#_Toc153873862)

[4.2 Разработка алгоритмов (описание алгоритмов по шагам, для двух методов) 13](#_Toc153873863)

[4.2.1 Алгоритм по шагам метода AddFileToZIP() 13](#_Toc153873864)

[4.2.2 Алгоритм по шагам метода ExtrOneFile() 13](#_Toc153873865)

[5 РЕЗУЛЬТАТ РАБОТЫ 15](#_Toc153873866)

[ЗАКЛЮЧЕНИЕ 19](#_Toc153873867)

[ЛИТЕРАТУРА 20](#_Toc153873868)

[ПРИЛОЖЕНИЕ А 21](#_Toc153873869)

[ПРИЛОЖЕНИЕ Б 22](#_Toc153873870)

[ПРИЛОЖЕНИЕ В 23](#_Toc153873871)

[ПРИЛОЖЕНИЕ Г 24](#_Toc153873872)

# ВВЕДЕНИЕ

В современном мире объем данных постоянно растет, и эффективное управление этим потоком информации становится все более актуальной задачей. Одним из ключевых инструментов в области обработки и хранения данных являются архиваторы, предоставляющие возможность компактного упаковывания и хранения информации. В рамках данного курсового проекта рассматривается создание архиватора, реализованного на языке программирования C++ с использованием библиотеки minizip, графический интерфейс которого был разработан с помощью фреймворка Qt в Qt Creator.

Архиваторы, также известные как упаковщики файлов, представляют собой программные инструменты, разработанные для сжатия и упаковки файлов с целью уменьшения их размера. Они играют ключевую роль в области управления данными, обеспечивая эффективное использование дискового пространства, ускоряя передачу информации по сети и облегчая хранение файлов. Изучение и разработка программного обеспечения для архивации предоставляют практические инструменты для облегчения хранения и обмена информацией.

Minizip – это небольшая, но мощная библиотека, которая построена на базе библиотеки zlib, предназначенная для работы с архивами формата ZIP.

Одним из ключевых преимуществ библиотеки Minizip является ее открытый исходный код, который позволяет разработчикам адаптировать библиотеку под свои потребности и внедрять ее в собственные проекты. Написанная на языке программирования C, Minizip обеспечивает высокую производительность и эффективное использование ресурсов, что особенно важно при работе с большими объемами данных.

Применение Minizip в разработке архиватора на языке программирования C++ предоставляет уникальную возможность не только ознакомиться с принципами работы ZIP-формата, но и легко интегрировать этот функционал в собственные проекты. Данная библиотека становится надежным инструментом для решения задач архивации данных, открывая перед разработчиками новые горизонты в области обработки и хранения информации.

Целью данного курсового проекта является не только создание функционального архиватора с графически интерфейсом, но и изучение основных принципов архитектуры программного обеспечения, эффективных методов сжатия данных, а также особенностей работы с библиотекой minizip в контексте разработки на языке программирования C++.

# 1 ПОСТАНОВКА ЗАДАЧИ

Тема курсового проекта – архиватор.

Программа должна:

* архивировать файлы.
* разархивировать без потери информации.
* иметь графический интерфейс для удобства пользователя.
* поддерживать работу с командной строкой.

# 2 ОБЗОР ЛИТЕРАТУРЫ

Архиваторы – это программы, предназначенные для упаковки и распаковки файлов с целью сжатия данных и экономии места на диске или их передачи по сети. Они работают путем объединения нескольких файлов в один архивный файл и применения различных методов сжатия для уменьшения его размера.

Вот несколько популярных архиваторов:

1. WinRAR: Это один из самых известных архиваторов под Windows. WinRAR поддерживает различные форматы архивов, такие как RAR, ZIP, и может работать с алгоритмами сжатия, такими как RAR, ZIP, GZIP, и другими.

2. 7-Zip: Это свободно распространяемый архиватор с открытым исходным кодом. 7-Zip поддерживает широкий спектр форматов, включая свой собственный 7z, который обеспечивает хорошую степень сжатия.

3. WinZip: Еще один популярный архиватор, преимущественно используемый в среде Windows. WinZip поддерживает множество форматов, включая ZIP, RAR, и другие.

4. PeaZip: Это бесплатный архиватор с открытым исходным кодом, который поддерживает большое количество форматов архивов, включая свой формат PeaZip, ZIP, RAR, и многие другие.

5. Tar и gzip: В мире Unix-подобных систем часто используется комбинация утилит tar для создания архивов и gzip для их сжатия. Gzip выполняет только две функции: сжатие и распаковку одного файла; упаковка нескольких файлов в один архив невозможна. При сжатии к оригинальному расширению файла добавляется суффикс .gz. Для упаковки нескольких файлов обычно их сначала архивируют (объединяют) в один файл утилитой Tar, а потом этот файл сжимают с помощью Gzip. Таким образом, сжатые архивы обычно имеют двойное расширение «.tar.gz», либо сокращённое «.tgz».

Эти архиваторы обеспечивают различные уровни сжатия, скорости работы и функциональности. Выбор конкретного архиватора зависит от ваших потребностей, предпочтений и операционной системы, которую вы используете. Существует огромное количество видов типов архивов. Рассмотрим некоторые из них:

Архиваторы используют множество методов и алгоритмов сжатия без потерь. Методы сжатия данных без потерь предназначены для уменьшения размера данных без потери какой-либо информации. Эти методы обеспечивают точное восстановление оригинальных данных из сжатого представления. Рассмотрим некоторые из них:

– LZ77 – метод использует словарь для замены повторяющихся фрагментов текста на ссылки на предыдущие вхождения. Когда обнаруживается повторение, создается токен, указывающий на предыдущее вхождение;

– LZ78 – метод строит словарь динамически, добавляя новые фразы, и кодирует фразы в виде пар (индекс, символ). Он используется в формате архивации Lempel-Ziv-Welch (LZW), применяемом в формате GIF;

– Huffman Coding – метод строит оптимальное бинарное дерево кодирования для символов на основе их частоты встречаемости. Чем чаще символ встречается, тем короче его код. Этот метод использует переменную длину кодирования для представления символов с разной частотой появления. Часто встречающиеся символы получают более короткие коды, что позволяет уменьшить общий размер файла. Данный широко применяется в архиваторах и сжатии текстовых данных;

– Arithmetic Coding – метод кодирует всё сообщение в виде одного числа в интервале [0, 1) на основе вероятностей встречаемости символов. Алгоритм разбивает интервал на подинтервалы, представляющие каждый символ. Arithmetic Coding обеспечивает более эффективное сжатие, чем Huffman Coding, но также более сложен для реализации;

– Burrows-Wheeler Transform (BWT) – метод изменяет порядок символов в сообщении, создавая блоки данных, где повторения символов становятся более вероятными. Далее применяется алгоритм Move-to-Front (MTF) и Run-Length Encoding (RLE) для дополнительного сжатия. BWT часто используется в алгоритмах сжатия данных, таких как BZIP2;

– Delta Encoding – метод сжатия основан на разнице между последовательными элементами данных. Если значения похожи, их разница может быть представлена более компактно. Он особенно полезен, например, при сжатии последовательности чисел;

– Run-Length Encoding (RLE) – Этот простой метод основан на представлении последовательностей одинаковых символов или байтов одним экземпляром и их количеством. Например, строка "AAAABBBCCDAA" может быть закодирована как "4A3B2C1D2A".

Эти методы могут применяться отдельно или в комбинациях для достижения наилучших результатов в зависимости от характеристик конкретных данных. Каждый метод имеет свои преимущества и недостатки, и их эффективность может сильно зависеть от характера данных, которые они обрабатывают.

## 2.1 Обзор методов и алгоритмов поставленной задачи

DEFLATE – это алгоритм сжатия данных без потерь, который широко применяется в формате архивации ZIP. Он был создан Филом Кэтцем и Джеймсом Лемпелем, и его основная идея заключается в использовании комбинации двух других алгоритмов сжатия данных - LZ77 и Huffman Coding.

Алгоритм DEFLATE работает следующим образом.

1. LZ77 (Lempel-Ziv 77) – DEFLATE начинается с применения алгоритма LZ77, который ищет повторяющиеся последовательности символов в данных. Когда обнаруживается повторение, алгоритм создает токен, который указывает на предыдущее вхождение этой последовательности. Таким образом, повторяющиеся блоки данных заменяются более короткими токенами.

2. Huffman Coding – полученные токены, а также остальные символы данных, кодируются с использованием алгоритма Huffman Coding. Этот шаг направлен на эффективное представление часто встречающихся символов более короткими битовыми кодами, что приводит к дополнительному уменьшению размера данных.

3. Словарь для Huffman Coding – далее алгоритм DEFLATE строит динамический словарь для Huffman Coding, основываясь на частоте встречаемости символов. Это позволяет адаптироваться к структуре данных и обеспечивать более эффективное сжатие.

4. Блочное сжатие – данные обрабатываются блоками, и для каждого блока строится свой собственный словарь Huffman Coding. Это позволяет DEFLATE лучше обрабатывать различные части данных и добиваться более эффективного сжатия.

DEFLATE успешно комбинирует преимущества обоих методов - LZ77 для обнаружения повторяющихся блоков и Huffman Coding для их эффективного кодирования. В результате DEFLATE стал одним из наиболее популярных методов сжатия данных и широко применяется в архиваторах, сетевых протоколах (например, HTTP), и других областях, где важно уменьшение размера передаваемых или хранимых данных.

Описанный выше алгоритм реализован в функционале библиотеки Minizip, которая построена на базе библиотеки Zlib.

Zlib – это библиотека с открытым исходным кодом, разработанная для обеспечения эффективного сжатия и распаковки данных. Название "zlib" происходит от слов "zippy" (быстрый) и "library" (библиотека), что иллюстрирует ключевую цель библиотеки – обеспечить высокую скорость работы с данными при их сжатии и распаковке.

Minizip интегрирует Zlib для выполнения операций сжатия и распаковки данных внутри ZIP-архивов. Таким образом, Minizip является более высокоуровневой библиотекой, которая использует Zlib как один из своих компонентов. Использование Zlib в Minizip позволяет достичь высокой эффективности сжатия данных в формате ZIP.

Основные возможности Minizip включают в себя создание ZIP-архивов, извлечение файлов из существующих архивов, добавление или удаление файлов из архива, а также множество опций для управления структурой и параметрами ZIP-файлов. Библиотека minizip поддерживает стандартные функции сжатия данных, такие как Deflate, что позволяет эффективно уменьшать объем архивируемой информации.

Графический интерфейс риложения реализован с помощью фреймворка Qt в IDE (интегрированная среда разработки) Qt Creator.

Qt – это кроссплатформенный фреймворк для разработки программного обеспечения. Он предоставляет набор инструментов и библиотек для создания графических интерфейсов пользователя (GUI), обработки событий, работы с сетью, базами данных, многопоточностью и другими аспектами разработки приложений.

В Qt используются два основных подхода к созданию GUI: использование виджетов и QML (Qt Meta-Object Language).

Виджеты – это основные строительные блоки графического интерфейса в Qt. Они представляют собой различные элементы управления, такие как кнопки, текстовые поля, списки и т. д. Qt предоставляет богатую библиотеку предопределенных виджетов, которые можно использовать для построения стандартных элементов интерфейса. Расположение виджетов в окне осуществляется с использованием различных менеджеров компоновки (layout managers), таких как QVBoxLayout, QHBoxLayout и QGridLayout, обеспечивая гибкость и адаптивность интерфейса.

QML (Qt Meta-Object Language) – это декларативный язык разметки, который позволяет создавать гибкие и анимированные интерфейсы. QML используется для описания структуры интерфейса и взаимодействия между его элементами. В QML элементы интерфейса называются компонентами, и они описываются с использованием синтаксиса, близкого к JavaScript. QML поддерживает анимации, трансформации, обработку событий и многие другие возможности для создания интересных и динамичных пользовательских интерфейсов.

Qt Creator – это интегрированная среда разработки (IDE) для работы с проектами, основанными на Qt. Она предоставляет редактор кода, отладчик, дизайнер интерфейсов и множество других инструментов для упрощения работы разработчиков и обеспечения эффективного процесса создания приложений.

# 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

## 3.1. Структура входных и выходных данных

Программа принимает в качестве входных данных строки, которые пользователь вводит через консоль или графический интерфейс, а также файлы для архивации или архив для разархивации.

Выходными данными программы является ZIP-архив или файлы, извлеченные из архива.

## 3.2. Разработка диаграммы классов

Диаграмма классов представлена в приложении А.

## 3.3. Описание классов

### 3.3.1 Класс ClassAddToZIP

ClassAddToZIP – класс для добавления в архив файла или содержимого папки.

Поля:

– string zipFilePath – хранит путь, по которому будет создан архив;

– string f\_Path – хранит путь к файлу или папке, который будет добавлен в архив;

– string zipFileName – хранит имя создаваемого архива;

­ string ziplevel – хранит степень сжатия создаваемого архива.

Методы:

+ void AddFolderToZip() – создает архив и добавляет содержимое папки в архив;

+ void AddFileToZip() – добавляет один файл в уже существующий архив;

+ void AddToZip() – определяет работу с папкой или файлом.

## **3.3.2** Класс ClassExtractZIP

ClassExtractZIP – класс для извлечения из архива одного файла или всего содержимого.

Поля:

– std::string ExtrToPath – хранит путь, по которому будет распакован архив;

– std::string z\_Path – хранит путь к архиву, который будет распакован;

– std::string ExtrFileName – хранит имя файла в архиве, который будет извлечен.

Методы:

+ void ExtrAll() – извлекает все содержимое архива;

+ void ExtrOneFile() – извлекает один файл из архива;

+ void ExtrZip() – определяет работу со всем содержимым архива или одним файлом.

### 3.3.3 Класс MainWindow

MainWindow – класс графического дисплея архиватора.

Поля:

– string \_f\_path – хранит путь к файлу или папке, который необходимо добавить в архив;

– string \_path\_add – хранит путь, по которому будет создан архив;

– string \_name\_zip – хранит имя создаваемого архива;

– string \_z\_path – хранит путь к архиву, который необходимо разархивировать;

– string \_path\_extr – хранит путь, по которому необходимо разархивировать архив;

– string \_name\_file – хранит имя файла в архиве или команду для извлечения всех файлов из архива;

– string \_level – хранит степень сжатия создаваемого архива.

Методы:

+ void on\_button\_add\_clicked() – отправка данных архиватору для архивации файлов или содержимого папки;

+ void on\_button\_add\_path\_clicked() – выбор пути, по которому будет создан архив;

+ void on\_button\_files\_clicked() – выбор файла для добавления в архив;

+ void on\_button\_folder\_clicked() – выбор папки для добавления в архив;

+ void on\_button\_z\_path\_clicked() – выбор архива для разархивации;

+ void on\_button\_extr\_to\_clicked() – выбор пути, по которому будет разархивирован архив;

+ void on\_button\_extr\_clicked() – отправка данных архиватору для разархивации архива;

+ void on\_minButton\_clicked() – выбор минимальной степени сжатия;

+ void on\_medButton\_clicked() – выбор средней степени сжатия;

+ void on\_maxButton\_clicked() – выбор максимальной степени сжатия;

+ void on\_noButton\_clicked() – выбор нулевой степени сжатия (без сжатия).

# 4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

## 4.1 Разработка схем алгоритмов (два наиболее важных метода)

Схема алгоритма метода AddToZIP() представлена в приложении Б, Схема алгоритма метода ExtrZIP() представлена в приложении В.

## 4.2 Разработка алгоритмов (описание алгоритмов по шагам, для двух методов)

### 4.2.1 Алгоритм по шагам метода AddFileToZIP()

Шаг 1. Проверка существования архива zipFileName по пути zipFilePath с помощью fs::exists, если он не существует, то используем флаг APPEND\_STATUS\_CREATE для создания архива, если существует – APPEND\_STATUS\_ADDINZIP для дозаписи в существующий архив.

Шаг 2. Создаем или открываем (в зависимости от флага) zipFileName с помощью функции zipOpen, если не удалось создать или открыть zipFileName, то выводим в консоль сообщение об ошибке и закрываем программу.

Шаг 3. Открываем файл с помощью fopen. Если не удалось открыть файл, то выводим в консоль сообщение об ошибке и закрываем программу.

Шаг 4. Создаем новый файл в архиве с помощью zipOpenNewFileInZip. Если не удалось создать файл, то закрываем файл с помощью fclose, выводим в консоль сообщение об ошибке и закрываем программу

Шаг 5. Записываем данные в созданный файл в архиве с помощью zipWriteInFileInZip в цикле do-while.

Шаг 6. Закрываем файл в архиве – zipCloseFileInZip.

Шаг 7. Закрываем файл – fclose.

Шаг 8. Закрываем архив – zipClose.

Шаг 9. Конец.

### 4.2.2 Алгоритм по шагам метода ExtrOneFile()

Шаг 1. Открываем архив z\_Path с помощью unzOpen. Если не удалось открыть файл, то выводим в консоль сообщение об ошибке и закрываем программу.

Шаг 2. Ищем необходимый файл в архиве с помощью функции unzLocateFile. Если файл найден, то функция возвращает UNZ\_OK, иначе закрываем архив с помощью unzClose, выводим в консоль сообщение об ошибке и закрываем программу

Шаг 3. Получаем данные о текущем файле в ZIP-архиве с помощью unzGetCurrentFileInfo, на который указывает указатель unzFile.

Шаг 4. Выводим в консоль путь куда будет извлечен файл.

Шаг 5. Создаем бинарный файл с помощью fopen по пути указанном в ExtrToPath.

Шаг 6. Открываем файл в архиве с помощью unzOpenCurrentFile.

Шаг 7. Читаем файл в архиве с помощью unzReadCurrentFile.

Шаг 8. Записываем прочитанные данные из файла в архиве в бинарный файл на диске с помощью fwrite в цикле do-while.

Шаг 9. Закрываем файл fclose.

Шаг 10. Закрываем архив в файле unzCloseCurrentFile.

Шаг 11. Закрываем архив unzClose.

Шаг 12. Конец.

# 5 РЕЗУЛЬТАТ РАБОТЫ

Интерфейс приложения состоит из одного окна. В верхней части окна расположены кнопки (вкладки) выбора режима работы приложения: «Архивация» – добавить в архив, «Разархивация» – извлечь из архива.

Рассмотрим подробнее вкладку «Архивация». На рисунке 5.1 представлено окно графического интерфейса архиватора с активной вкладкой «Архивация». В окне находятся семь кнопок и одно поле для ввода текстовой строки.

Кнопки:

– кнопки выбора файла или папки для добавления в архив;

– кнопка выбора пути, по которому будет создан архив;

– четыре кнопки выбора степени сжатия;

– кнопка создания архива со всеми заданными выше параметрами.

Поле ввода – имя создаваемого архива.

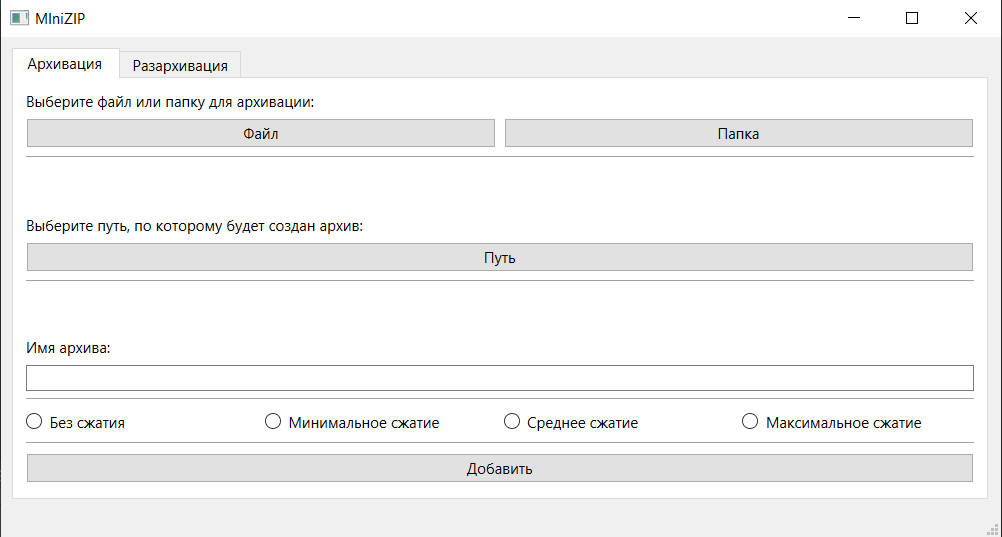


Рисунок 5.1 – Вкладка «Архивация»

Если при создании архива произойдет какая-либо ошибка и архив не будет создан, то на экране появится соответствующее окно с сообщением об ошибке. Окно с сообщением об ошибке приведено на рисунке 5.2.

Недопустимо, чтобы пути архивируемой папки и папки, в которой будет создан архив совпадали. В данном случае на экран будет выведено соответствующее предупреждение, а затем окон об ошибке. Окно с предупреждением приведено на рисунке 5.3.

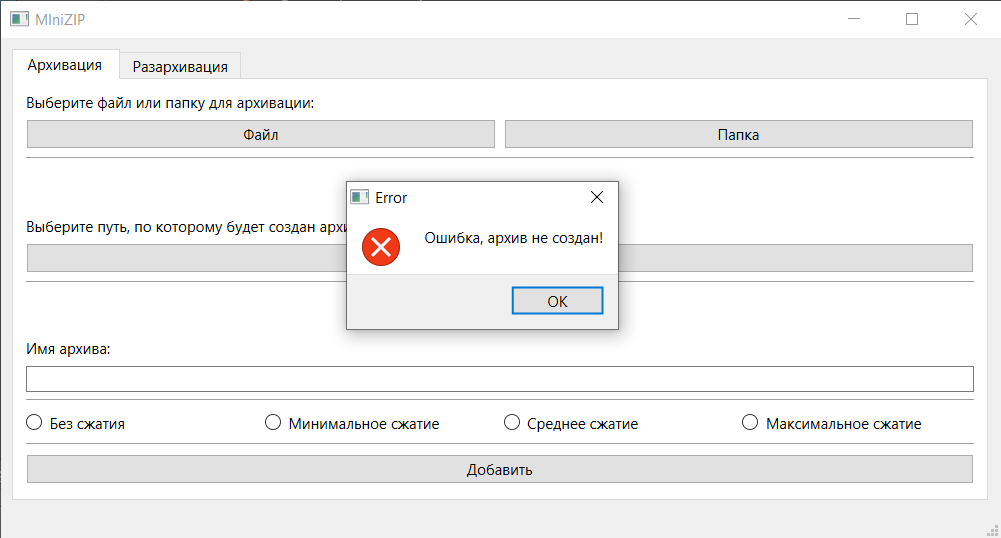


Рисунок 5.2 – Сообщение об ошибке

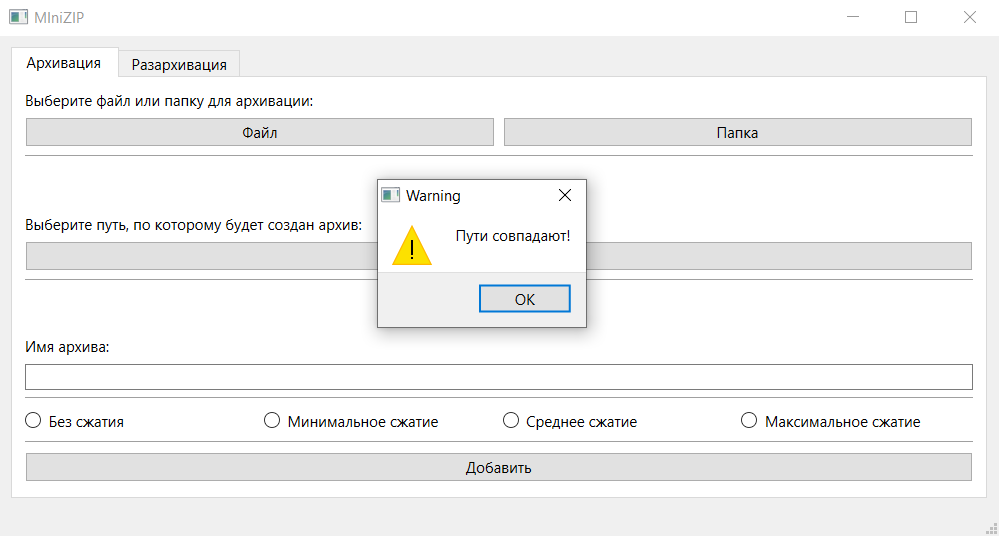


Рисунок 5.3 – Предупреждение о совпадающих путях

Когда архив будет создан, на экран будет выведено сообщение о том, что архив успешно создан и будет открыта директория в файловом проводнике, в которую был создан архив. Сообщение о созданном архиве приведено на рисунке 5.4.

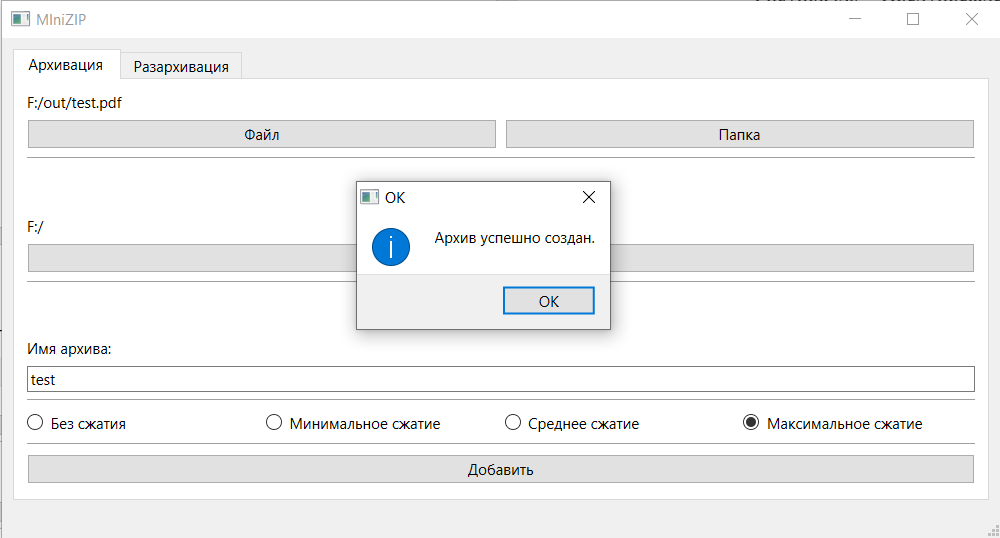


Рисунок 5.4 – Сообщение о созданном архиве

Рассмотрим подробнее вкладку «Разархивация». На рисунке 5.5 представлено окно графического интерфейса архиватора с активной вкладкой «Разархивация». В окне находятся три кнопки и одно поле для ввода.

Кнопки:

– кнопка выбора архива;

– кнопка выбора пути, по которому архив будет распакован;

– кнопка распаковки архива с заданными параметрами.

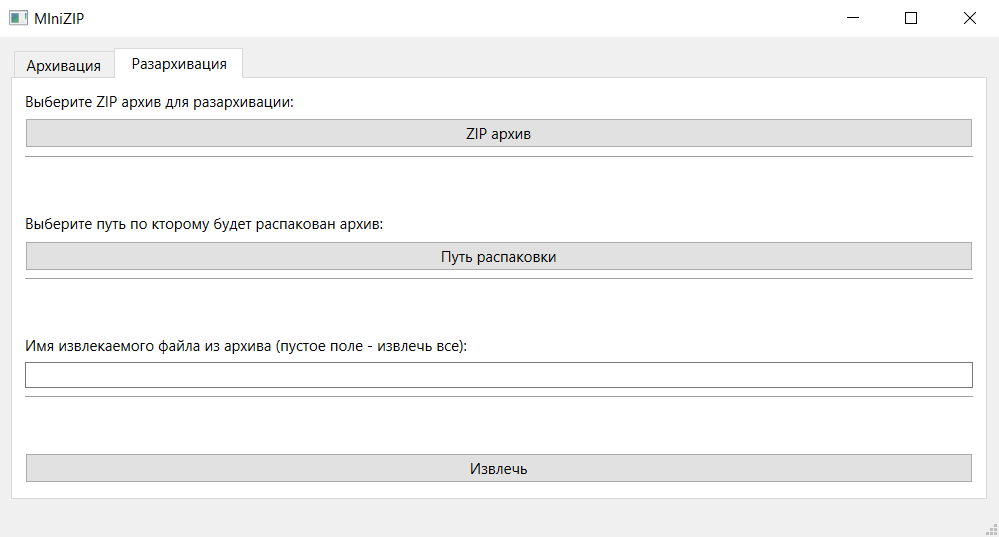


Рисунок 5.5 – Вкладка «Разархивация»

Поле ввода – имя файла в архиве, который необходимо извлечь, если необходимо извлечь все файлы из архива, то нужно оставить поле пустым.

Если файлы из архива были успешно извлечены, то будет открыта директория в файловом проводнике, в которую были извлечены файлы.

Помимо графического режима работы, приложение поддерживает работу с командной строкой. Для работы с программой необходимо перейти в директорию, в которой она расположена. Далее необходимо ввести команду вида:

«Режим работы» «путь к файлу» «конечный путь» «имя» «сжатие»,

где «Режим работы» – «Add» – добавить в архив, «Extr» – извлечь из архива;

«путь к файлу» – путь к файлу (папке) или архиву, в зависимости от режима работы;

«конечный путь» – путь, по которому будет создан архив или извлечены файлы;

«имя» – имя созданного архива, извлеченного файла или команда извлечения всего архива;

«сжатие» – целое число – уровень сжатия архива, допустимые значения: «-1» – среднее (стандартное) сжатие, «0» – без сжатия, «1» – минимальное (быстрое) сжатие, «9» – максимальное сжатие.

Пример работы программы в режиме «Add» с помощью командной строки приведен на рисунке 5.6.



Рисунок 5.6 – Пример работы программы режиме «Add» с командной строкой

Пример работы программы в режиме «Extr» с помощью командной строки приведен на рисунке 5.7.

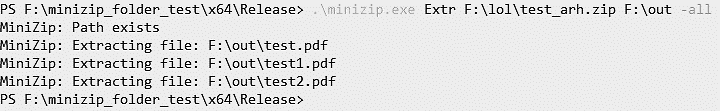


Рисунок 5.6 – Пример работы программы в режиме «Extr» с командной строкой

# ЗАКЛЮЧЕНИЕ

Архиваторы, как ключевые инструменты в области обработки и управления данными, продолжают оставаться неотъемлемой частью современного цифрового мира. Разработанные с целью оптимизации хранения и передачи информации, они обеспечивают высокий уровень эффективности и удобства для конечных пользователей.

Реализация архиватора на языке программирования C++ с использованием библиотеки minizip подчеркивают значимость применения современных технологий в области архивации данных. Этот проект не только предоставляют возможность глубже понять принципы алгоритмов сжатия, но и являются практическим примером интеграции современных методов обработки информации.

Создание архиватора на основе библиотеки minizip расширяет знания в области архитектуры программного обеспечения, эффективного управления ресурсами и применения алгоритмов сжатия данных.

В свете постоянного роста объема данных и потребности в их эффективном управлении, архиваторы остаются неотъемлемой частью цифровой инфраструктуры. Развитие и совершенствование подобных программных продуктов будет продолжаться, внося свой вклад в улучшение процессов хранения, обмена и обработки данных в цифровой эпохе.

# ЛИТЕРАТУРА

[1] GitHub: Библиотека Minizip [Электронный ресурс]. – Режим доступа: https://github.com/zlib-ng/minizip-ng. – Дата доступа: 25.10.2023.

[2] GitHub: Библиотека Zlib [Электронный ресурс]. – Режим доступа: https://github.com/zlib-ng. – Дата доступа: 25.10.2023.

[3] Wikipedia: Архиватор [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/File\_archiver. – Дата доступа: 25.10.2023.

[4] Компьютер пресс: Сравнение 64-битных архиваторов WinRAR 4.2, WinZip 17.0 и 7-Zip 9.30 [Электронный ресурс] – Режим доступа: https://compress.ru/article.aspx?id=23664#12. – Дата доступа: 26.10.2023.

[5] Qt Documentation [Электронный ресурс] – Режим доступа: https://doc.qt.io. – Дата доступа: 01.11.2023.

# ПРИЛОЖЕНИЕ А

*(обязательное)*

Диаграмма классов

# ПРИЛОЖЕНИЕ Б

*(обязательное)*

Схема алгоритма метода AddToZIP()

# ПРИЛОЖЕНИЕ В

*(обязательное)*

Схема алгоритма метода ExtrZIP()

# ПРИЛОЖЕНИЕ Г

*(обязательное)*

Исходный текст программы

**main.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "ClassAddToZip.h"

#include "ClassExtractZip.h"

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

{

const std::string action = argv[1];

if (action == "Add") {

const std::string f\_Path = argv[2];

const std::string zipFilePath = argv[3];

const std::string zipFileName = argv[4];

const std::string ziplevel = argv[5];

ClassAddToZIP addfolder(f\_Path, zipFilePath, zipFileName, ziplevel);

addfolder.AddToZip();

}

else if (action == "Extr") {

const std::string z\_Path = argv[2];

const std::string ExtrToPath = argv[3];

const std::string FileName = argv[4];

ClassExtractZip extruct(z\_Path, ExtrToPath, FileName);

extruct.ExtrZip();

}

else if (action == "List") {

const std::string z\_Path = argv[2];

const std::string ExtrToPath = "-list";

const std::string FileName = "list";

ClassExtractZip extruct(z\_Path, ExtrToPath, FileName);

extruct.ExtrList();;

}

else {

std::cout << "MiniZip: incorrect input" << std::endl;

return 1;

}

return 0;

}

**ClassAddToZIP.h**

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <minizip/zip.h>

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <filesystem>

#include <cstdio> // Для FILE

#include <cstring> // Для std::string.c\_str()

namespace fs = std::filesystem;

class ClassAddToZIP

{

public:

ClassAddToZIP(const std::string& \_f\_Path, const std::string& \_zipFilePath, const std::string& \_zipFileName) {

zipFilePath = \_zipFilePath;

f\_Path = \_f\_Path;

zipFileName = \_zipFileName;

}

~ClassAddToZIP() {};

void AddFolderToZip();

void AddFileToZip();

void AddToZip();

protected:

std::string zipFilePath;

std::string f\_Path;

std::string zipFileName;

};

**ClassExtractZIP.h**

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <minizip/unzip.h>

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <filesystem>

#include <cstdio> // Для FILE

#include <cstring> // Для std::string.c\_str()

namespace fs = std::filesystem;

class ClassExtractZip

{

public:

ClassExtractZip(const std::string& \_z\_Path, const std::string& \_ExtrToPath, const std::string& \_ExtrFileName) {

ExtrToPath = \_ExtrToPath;

z\_Path = \_z\_Path;

ExtrFileName = \_ExtrFileName;

};

~ClassExtractZip() {};

void ExtrAll();

void ExtrOneFile();

void ExtrZip();

void ExtrList();

protected:

std::string ExtrToPath;

std::string z\_Path;

std::string ExtrFileName;

};

**Add\_f.cpp**

#include "ClassAddToZip.h"

void ClassAddToZIP::AddToZip() {

fs::path path\_to\_check(f\_Path);

fs::path path\_to\_check2(zipFilePath);

if (fs::exists(path\_to\_check2) == true) {

std::cout << "MiniZip: Path exists - OK" << std::endl;

}

else {

fs::create\_directories(zipFilePath);

std::cout << "MiniZip: Path created - OK" << std::endl;

}

if (fs::exists(path\_to\_check)) {

if (std::filesystem::is\_directory(path\_to\_check)) {

ClassAddToZIP::AddFolderToZip();

}

else {

ClassAddToZIP::AddFileToZip();

}

}

else {

std::cout << "MiniZip: ERROR File or folder is not exists!" << std::endl;

std::abort();

}

}

void ClassAddToZIP::AddFolderToZip()

{

int level = std::stoi(ziplevel);

if (level < -1 || level > 1 || level != 9)

level = -1;

zipFile zip = zipOpen((zipFilePath + "\\" + zipFileName).c\_str(), APPEND\_STATUS\_CREATE);

if (!zip)

{

std::cerr << "MiniZip: ERROR in AddFolderToZip() - Failed to open ZIP: " << zipFileName << std::endl;

std::abort();

}

// Обходим все файлы в папке и добавляем их в архив

for (const auto& entry : fs::directory\_iterator(f\_Path))

{

const std::string filePath = entry.path().string();

const std::string entryName = entry.path().filename().string();

// Открываем файл для чтения

FILE\* file = fopen(filePath.c\_str(), "rb");

if (!file)

{

std::cerr << "MiniZip: ERROR in AddFolderToZip() - Failed to open file: " << filePath << std::endl;

std::abort();

}

// Открываем новый файл в архиве

zip\_fileinfo zipFileInfo = {};

int result = zipOpenNewFileInZip(zip, entryName.c\_str(), &zipFileInfo, nullptr, 0, nullptr, 0, nullptr, Z\_DEFLATED, level);

if (result != ZIP\_OK)

{

std::cerr << "MiniZip: ERROR in AddFolderToZip() - Failed to open file in ZIP: " << filePath << std::endl;

fclose(file);

std::abort();

}

// Записываем данные в файл в архиве

const int bufferSize = 4096;

char buffer[bufferSize];

size\_t bytesRead;

do {

bytesRead = fread(buffer, 1, bufferSize, file);

if (bytesRead > 0)

{

zipWriteInFileInZip(zip, buffer, static\_cast<unsigned int>(bytesRead));

}

} while (bytesRead > 0);

// Закрываем файл в архиве

zipCloseFileInZip(zip);

fclose(file);

}

zipClose(zip, nullptr);

}

void ClassAddToZIP::AddFileToZip() {

fs::path path\_to\_checkZIP(zipFilePath + "\\" + zipFileName);

int status\_addzip;

if (fs::exists(path\_to\_checkZIP) == true)

status\_addzip = APPEND\_STATUS\_ADDINZIP;

else

status\_addzip = APPEND\_STATUS\_CREATE;

int level = std::stoi(ziplevel);

if (level < -1 || level > 1 || level != 9)

level = -1;

zipFile zip = zipOpen((zipFilePath + "\\" + zipFileName).c\_str(), status\_addzip);

if (!zip)

{

std::cerr << "MiniZip: ERROR in AddFileToZip() - Failed to open ZIP: " << path\_to\_checkZIP << std::endl;

std::abort();

}

fs::path filePath = f\_Path;

std::string fileName = filePath.filename().string();

// Открываем файл для чтения

FILE\* file = fopen(f\_Path.c\_str(), "rb");

if (!file)

{

std::cerr << "MiniZip: ERROR in AddFileToZip() - Failed to open file: " << filePath << std::endl;

std::abort();

}

// Открываем новый файл в архиве

zip\_fileinfo zipFileInfo = {};

int result = zipOpenNewFileInZip(zip, fileName.c\_str(), &zipFileInfo, nullptr, 0, nullptr, 0, nullptr, Z\_DEFLATED, level);

if (result != ZIP\_OK)

{

std::cerr << "MiniZip: ERROR in AddFileToZip() - Failed to open file in ZIP: " << filePath << std::endl;

fclose(file);

std::abort();

}

// Записываем данные в файл в архиве

const int bufferSize = 4096;

char buffer[bufferSize];

size\_t bytesRead;

do {

bytesRead = fread(buffer, 1, bufferSize, file);

if (bytesRead > 0)

{

zipWriteInFileInZip(zip, buffer, static\_cast<unsigned int>(bytesRead));

}

} while (bytesRead > 0);

zipCloseFileInZip(zip);

fclose(file);

zipClose(zip, nullptr);

}

**Extr\_f.cpp**

#include "ClassExtractZip.h"

void ClassExtractZip::ExtrZip() {

fs::path path\_to\_check(ExtrToPath);

if (fs::exists(path\_to\_check) == true) {

std::cout << "MiniZip: Path exists" << std::endl;

}

else {

fs::create\_directories(ExtrToPath);

}

if (ExtrFileName == "-all")

ExtrAll();

else

ExtrOneFile();

}

void ClassExtractZip::ExtrAll() {

unzFile zip = unzOpen(z\_Path.c\_str());

if (!zip) {

std::cerr << "MiniZip: ERROR Failed to open the ZIP file." << std::endl;

return;

}

int result = unzGoToFirstFile(zip);

if (result != UNZ\_OK) {

std::cerr << "MiniZip: ERROR Failed to go to the first file in the ZIP archive." << std::endl;

unzClose(zip);

return;

}

do {

char fileName[256];

unz\_file\_info fileInfo;

result = unzGetCurrentFileInfo(zip, &fileInfo, fileName, sizeof(fileName), nullptr, 0, nullptr, 0);

if (result != UNZ\_OK) {

std::cerr << "MiniZip: ERROR Failed to get current file info." << std::endl;

break;

}

std::string outputPath = ExtrToPath + "\\" + fileName;

// This is a file, extract it

std::cout << "MiniZip: Extracting file: " << outputPath << std::endl;

//std::cout << "!!!: " << ExtrFileName << std::endl;

FILE\* outFile = fopen(outputPath.c\_str(), "wb");

if (!outFile) {

std::cerr << "MiniZip: ERROR Failed to open output file for writing." << std::endl;

break;

}

char buffer[4096];

int bytesRead;

unzOpenCurrentFile(zip);

do {

bytesRead = unzReadCurrentFile(zip, buffer, sizeof(buffer));

if (bytesRead > 0) {

fwrite(buffer, 1, bytesRead, outFile);

}

} while (bytesRead > 0);

fclose(outFile);

unzCloseCurrentFile(zip);

result = unzGoToNextFile(zip);

} while (result == UNZ\_OK && result != UNZ\_END\_OF\_LIST\_OF\_FILE);

unzClose(zip);

}

void ClassExtractZip::ExtrOneFile() {

unzFile zip = unzOpen(z\_Path.c\_str());

if (!zip) {

std::cerr << "MiniZip: ERROR Failed to open the ZIP file." << std::endl;

return;

}

int result = unzLocateFile(zip, ExtrFileName.c\_str(), 1); // 1 - case-sensitive, 0 - case-insensitive

if (result != UNZ\_OK) {

std::cerr << "MiniZip: ERROR Failed to go to the first file in the ZIP archive." << std::endl;

unzClose(zip);

return;

}

char fileName[256];

unz\_file\_info fileInfo;

result = unzGetCurrentFileInfo(zip, &fileInfo, fileName, sizeof(fileName), nullptr, 0, nullptr, 0);

std::string outputPath = ExtrToPath + "\\" + fileName;

// This is a file, extract it

std::cout << "MiniZip: Extracting file: " << outputPath << std::endl;

FILE\* outFile = fopen(outputPath.c\_str(), "wb");

char buffer[4096];

int bytesRead;

unzOpenCurrentFile(zip);

do {

bytesRead = unzReadCurrentFile(zip, buffer, sizeof(buffer));

if (bytesRead > 0) {

fwrite(buffer, 1, bytesRead, outFile);

}

} while (bytesRead > 0);

fclose(outFile);

unzCloseCurrentFile(zip);

unzClose(zip);

}

void ClassExtractZip::ExtrList() {

unzFile zip = unzOpen(z\_Path.c\_str());

if (!zip) {

std::cerr << "MiniZip: ERROR Failed to open the ZIP file." << std::endl;

return;

}

int result = unzGoToFirstFile(zip);

if (result != UNZ\_OK) {

std::cerr << "MiniZip: ERROR Failed to go to the first file in the ZIP archive." << std::endl;

unzClose(zip);

return;

}

int number\_file = 0;

do {

char fileName[256];

unz\_file\_info fileInfo;

result = unzGetCurrentFileInfo(zip, &fileInfo, fileName, sizeof(fileName), nullptr, 0, nullptr, 0);

if (result != UNZ\_OK) {

std::cerr << "MiniZip: ERROR Failed to get current file info." << std::endl;

break;

}

number\_file++;

std::cout << "MiniZip: List " << z\_Path << " " << number\_file << ") " << fileName << std::endl;

char buffer[4096];

int bytesRead;

unzOpenCurrentFile(zip);

do {

bytesRead = unzReadCurrentFile(zip, buffer, sizeof(buffer));

} while (bytesRead > 0);

unzCloseCurrentFile(zip);

result = unzGoToNextFile(zip);

} while (result == UNZ\_OK && result != UNZ\_END\_OF\_LIST\_OF\_FILE);

unzClose(zip);

}

**Mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QFileDialog>

#include <QMainWindow>

#include <QDebug>

#include <filesystem>

#include <QMessageBox>

#include <QDesktopServices>

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void on\_butt\_add\_clicked();

void on\_butt\_add\_path\_clicked();

void on\_butt\_files\_clicked();

void on\_butt\_folder\_clicked();

void on\_butt\_z\_path\_clicked();

void on\_butt\_extr\_to\_clicked();

void on\_butt\_extr\_clicked();

void on\_minButton\_clicked();

void on\_medButton\_clicked();

void on\_maxButton\_clicked();

void on\_noButton\_clicked();

private:

Ui::MainWindow \*ui;

std::string \_f\_path;

std::string \_path\_add;

std::string \_name\_zip;

std::string \_z\_path;

std::string \_path\_extr;

std::string \_name\_file;

std::string level;

};

#endif // MAINWINDOW\_H

**Main.cpp**

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

**MainWindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_butt\_add\_clicked()

{

std::string executablePath = "F:\\minizip\_folder\_test\\x64\\Release\\minizip.exe";

QString name\_zip = ui->line\_namezip->text();

\_name\_zip = name\_zip.toStdString();

// Аргументы для передачи в программу

std::string arg1 = "Add";

std::string arg2 = \_f\_path;

std::string arg3 = \_path\_add;

std::string name = \_name\_zip;

std::string zip = ".zip";

std::string command;

name.append(zip);

std::string arg4 = name;

std::string arg5 = level;

if (arg2 == arg3){

QMessageBox::warning(this, "Warning", "Пути совпадают!");

}

else {

std::string command = executablePath + " " + arg1 + " " + arg2 + " " + arg3 + " " + arg4 + " " + arg5;

std::system(command.c\_str());

}

std::string path = arg3 + "\\" + name;

// Проверка статуса

if ( std::filesystem::exists(path)){

QMessageBox::information(this, "OK", "Архив успешно создан.");

}

else {

QMessageBox::critical(this, "Error", "Ошибка, архив не создан!");

}

QString path\_to = QString::fromStdString(arg3);

// Открываем проводник файлов для указанной директории

QDesktopServices::openUrl("file:///" + path\_to);

}

void MainWindow::on\_butt\_add\_path\_clicked()

{

QString directoryPath = QFileDialog::getExistingDirectory(nullptr, "Выберите папку", QDir::homePath());

qDebug() << directoryPath;

ui->lbl\_add\_path->setText(directoryPath);

\_path\_add = directoryPath.toStdString();

}

void MainWindow::on\_butt\_files\_clicked()

{

QString fileName = QFileDialog::getOpenFileName(nullptr, "Выберите файл", QDir::homePath(), "Все файлы (\*)");

qDebug() << fileName;

ui->lbl\_f\_path->setText(fileName);

\_f\_path = fileName.toStdString();

}

void MainWindow::on\_butt\_folder\_clicked()

{

QString directoryPath = QFileDialog::getExistingDirectory(nullptr, "Выберите папку", QDir::homePath());

qDebug() << directoryPath;

ui->lbl\_f\_path->setText(directoryPath);

\_f\_path = directoryPath.toStdString();

}

void MainWindow::on\_butt\_z\_path\_clicked()

{

QString fileName = QFileDialog::getOpenFileName(nullptr, "Выберите архив", QDir::homePath(), "ZIP (\*.zip)");

qDebug() << fileName;

ui->lbl\_z\_path->setText(fileName);

\_z\_path = fileName.toStdString();

}

void MainWindow::on\_butt\_extr\_to\_clicked()

{

QString directoryPath = QFileDialog::getExistingDirectory(nullptr, "Выберите папку", QDir::homePath());

qDebug() << directoryPath;

ui->lbl\_extr\_to->setText(directoryPath);

\_path\_extr = directoryPath.toStdString();

}

void MainWindow::on\_butt\_extr\_clicked()

{

std::string executablePath = "F:\\minizip\_folder\_test\\x64\\Release\\minizip.exe";

QString name\_file = ui->line\_namefile->text();

\_name\_file = name\_file.toStdString();

// Аргументы для передачи в программу

std::string arg1 = "Extr";

std::string arg2 = \_z\_path;

std::string arg3 = \_path\_extr;

std::string arg4 = \_name\_file;

if (arg4 == ""){

arg4 = "-all";

}

// Формируем команду для запуска программы с аргументами

std::string command = executablePath + " " + arg1 + " " + arg2 + " " + arg3 + " " + arg4;

std::system(command.c\_str());

QString path = QString::fromStdString(arg3);

// Открываем проводник файлов для указанной директории

QDesktopServices::openUrl("file:///" + path);

}

void MainWindow::on\_minButton\_clicked()

{

level = "1";

}

void MainWindow::on\_medButton\_clicked()

{

level = "-1";

}

void MainWindow::on\_maxButton\_clicked()

{

level = "9";

}

void MainWindow::on\_noButton\_clicked()

{

level = "0";

}