Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему
АРХИВАТОР ФАЙЛОВ И ДИРЕКТОРИЙ
БГУИР КР 1-40 02 01 314 ПЗ

Студент: Латфулин В. Р.

Руководитель: Глоба А. А.

Минск 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3
1 ОБЗОР ЛИТЕРАТУРЫ 4
1.1 Обзор используемой литературы 4
1.2 Обзор предметной области 4
1.3 Анализ существующих аналогов 5
2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 8
2.1 Постановка задачи 8
2.2 Моделирование программы 8
2.2.1 Модуль пользовательского интерфейса 9
2.2.2 Модуль работы с приложением 9
2.2.3 Модуль выбора алгоритма 9
2.2.4 Модуль архивации и разархивации 9
3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 10
3.1 MainWindow 10
3.2 AddWindow 11
3.3 ExtractWindow 12
3.4 CompressosRLE 13
3.5 CompressosHuffman 13
3.6 Node 14
4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 15
4.1 Архивация файлов и директорий алгоритмом RLE compressRLE 16
4.2 Разархивация архива алгоритмом RLE decompressRLE 16
4.3 Нахождение всех вложенных файлов и директорий dirWalk 16
4.4 Получение относительного пути getRelativePath 17
5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ 18
6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 20
ЗАКЛЮЧЕНИЕ 24
ЛИТЕРАТУРА 25
ПРИЛОЖЕНИЕ А 26
ПРИЛОЖЕНИЕ Б 27
ПРИЛОЖЕНИЕ В 28

ВВЕДЕНИЕ

В современное время архивы играют важную роль в хранении данных. Они позволяют хранить наборы разнообразных файлов. Это достаточно удобно, например, для передачи данных – легче передавать один файл, чем несколько. Архивы также позволяют хранить информацию в удобной форме.

Запакованные (сжатые) файлы занимают меньше дискового пространства и быстрее передаются на другие компьютеры, чем обычные файлы. Работа со сжатыми файлами и папками в Windows аналогична работе с обычными файлами и папками. Объединив несколько файлов в одну сжатую папку, вы сможете с легкостью поделиться ими.

С момента возникновения первой программы (архиватор) данного типа, было написано очень много различных архиваторов, поддерживающих различные форматы архивов. Самым распространенным был ARJ, на втором месте почти сразу за ним ZIP, затем, следовали такие архиваторы, как ARC, ACE, LZH. Теперь ситуация изменилась. Первое место среди форматов архиваторов занимает ZIP, забрав его у ARJ, который отошел теперь на задний план, на втором месте RAR и потом следуют ACE, ARJ и другие менее популярные форматы.

RAR[7] – формат был разработан Евгением Рошалем, автором одноименного архиватора и благодаря удобному интерфейсу одновременно с хорошим сжатием завоевал популярность.

ZIP[8] – формат был разработан компанией PKWARE.

1 ОБЗОР ЛИТЕРАТУРЫ

Теперь немного о сжатии. Сжатие файлов зависит от используемого алгоритма и от содержания в файлах данных. Текстовые файлы, как правило, сжимаются очень хорошо, бинарные файлы – хуже, а файлы, содержимое которых уплотнено максимально (аудио-, видео-, а также программы – инсталляторы) – сжимаются хуже всех.

1.1 Обзор используемой литературы

Что представляет собой файл – это последовательность байт, которая оперирует процесс, открывший файл (ресурс).

Работа с файлами реализуется как правило с применением файловых систем, обеспечивающих организацию работы с файлами и абстракцию над носителями информации.

Архиватор — это программа для упаковывания и/или сжатия файлов в один пакет, а также их извлечения.[9]

Распаковка архивов выполняется с помощью того же архиватора. Большинство современных архиваторов также выполняют сжатие упаковываемых в архив данных.

Сжатие данный (*data compression***)** – это алгоритм преобразования данных, который производится с целью уменьшения занимаемого ими объёма. Обратная процедура называется восстановлением данных (распаковкой или декомпрессией).[10]

1.2 Обзор предметной области

Основное применение сжатия – это устранение избыточности, которая содержится в исходных данных. Примером избыточности является повторение фрагментов в тексте. Такого рода избыточность устраняется заменой повторяющихся последовательности ссылкой на уже закодированный фрагмент с указанием его длины.

Существующие алгоритмы сжатия данных можно разделить на два больших класса – без потерь, и с потерями данных[11]:

- Алгоритмы сжатия без потерь применяются для уменьшения размера данных, и работают таким образом, что возможно восстановить данные в точности такими, какие они были до сжатия. Они применяются в коммуникациях, архиваторах и некоторых алгоритмах сжатии аудио и графической информации.
- Алгоритмы с потерями обычно применяются для сжатия изображений и аудио. Эти алгоритмы позволяют достичь больших степеней сжатия благодаря избирательной потере качества. Однако, по определению,

восстановить первоначальные данные из сжатого результата невозможно.

Основной принцип алгоритмов сжатия базируется на том, что в любом файле, содержащем неслучайные данные, информация которых полностью или частично повторяется.

Для частичных повторений, используя статистические математические модели можно определить вероятность повторения определённой комбинации символов. После этого можно создать коды, обозначающие выбранные фразы, и назначить самым часто повторяющимся фразам самые короткие коды. Для этого используются разные техники, например, *энтропийное кодирование*, кодирование повторов, и сжатие при помощи словаря. С их помощью 8-битный символ, или целая строка, могут быть заменены всего лишь несколькими битами, устраняя таким образом излишнюю информацию.

1.3 Анализ существующих аналогов

WinRAR

WinRAR - архиватор файлов для 32- и 64-разрядных операционных систем Windows (также существуют или существовали версии для Android, Linux, FreeBSD, macOS, MS-DOS), позволяющий создавать, изменять и распаковывать архивы RAR и ZIP, а также работать с множеством архивов других форматов.

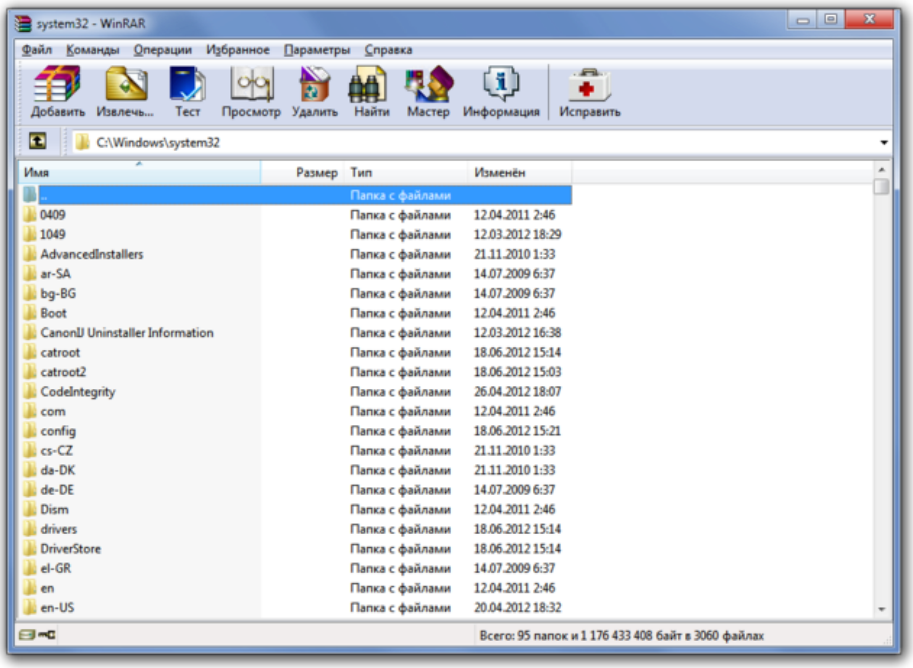

Рисунок 1.1 — Скриншот программы WinRAR

Автором WinRAR является Евгений Рошал. Приложение было написано на языке C++ и активно обновляется.

7-ZIP

7-Zip — свободный файловый архиватор с высокой степенью сжатия данных. Поддерживает несколько алгоритмов сжатия и множество форматов данных, включая собственный формат 7z с высокоэффективным алгоритмом сжатия LZMA. Программа разрабатывается с 1999 года, она бесплатна и имеет открытый исходный код. При сжатии в формате 7z также используются специальные фильтры-нормализаторы. Так, для более оптимального сжатия 32-битного x86-кода используются нормализующие конвертеры BCJ и BCJ2. Кроме того, программа имеет оптимизирующий дельта-конвертер для некоторых типов мультимедийных данных, например несжатых 24-битных изображений.

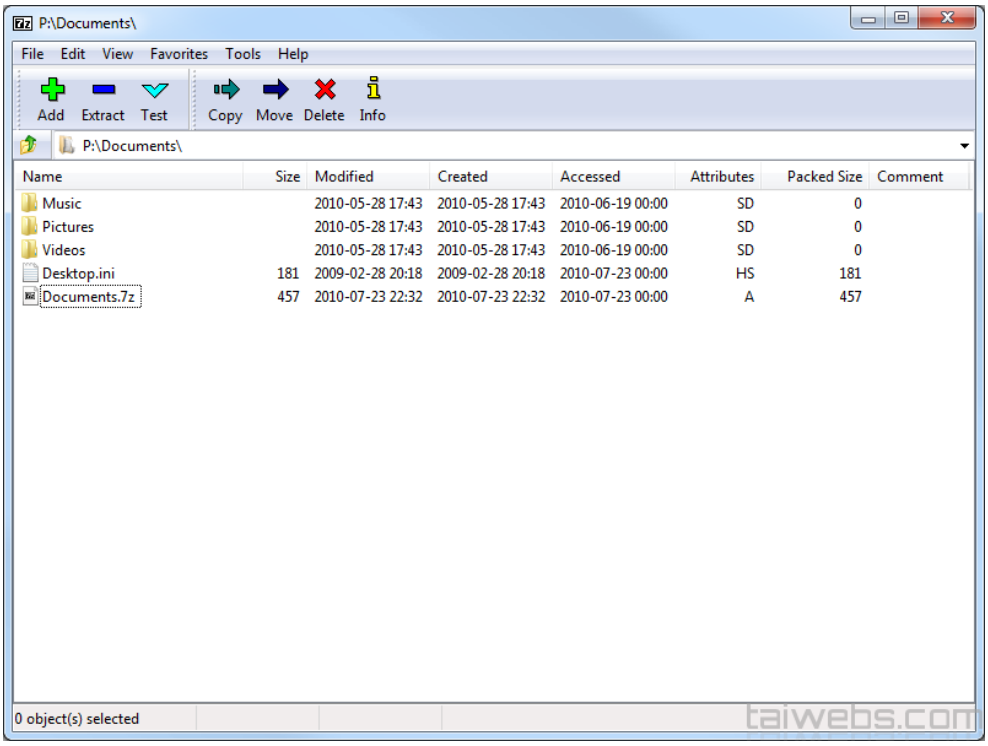

Рисунок 1.3 — Скриншот программы 7-Zip

7-zip написан на C++ и C. Первый выпуск 18 июля 1999 и активно используется и поддерживается по сегодняшний день.

PeaZip

PeaZip — свободный(GNU Lesser General Public License) и бесплатный кроссплатформенный архиватор и графическая оболочка для других архиваторов.

PeaZip поддерживает собственный формат архивов Pea (с поддержкой сжатия, многотомных архивов и гибкой системы шифрования и контроля целостности) и другие форматы, используя для многих из них внешние программы и библиотеки.

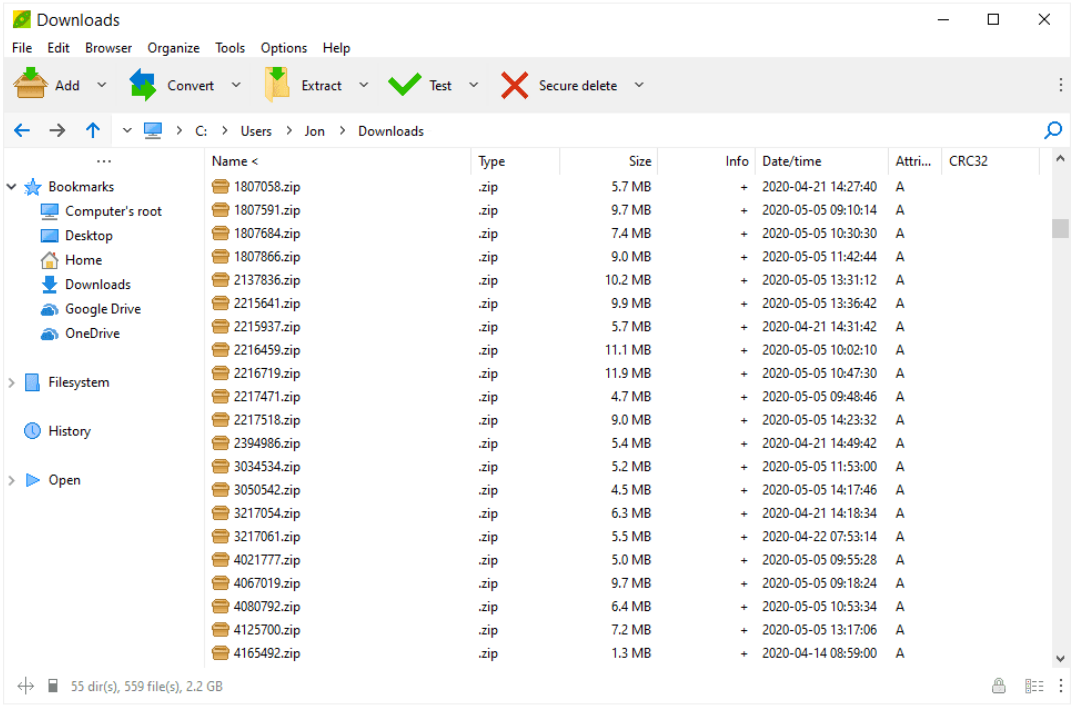

Рисунок 1.3 — Скриншот программы PeaZip

PearZip был выпущен 16 сентября 2006, а последняя версия – 29 ноября 2012).

Алгоритмы сжатия в данном курсовом проекте легко реализовать на базе полученных знаний за предыдущее время обучение и за такой промежуток времени по сравнению с другими алгоритмами сжатия. Также есть и другие алгоритмы сжатия данных:

- zip.
- rar.
- winrar.
- pkzip.
- gzip.
- bzip2.

Данные алгоритмы сжатия находят свое применение в более крупных аналоговых проектах.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Для начала следует поставить функциональные требования разрабатываемой программе. Разбить утилиту на модули и функциональные блоки. При применении данного подхода упрощается понимание проектирования проекта. После того как проведен краткий экскурс в основные понятия, можно обозначить цели для выполнения разрабатываемого продукта.

2.1 Постановка задачи

Программа должна иметь удобный пользовательский интерфейс с командами, которые понадобятся пользователю для удобной работы с файлами. Все команды должны быть полезными и раскрывать весь потенциал программы. В качестве языка программирования выбран C++, по причине быстрой производительности, требуемой для обработки большого количества объектов и наличия опыта в использования данного языка.

В качестве реализация графического интерфейса используется фреймворк Qt, основанный на языке C++. Qt обладает большим количеством базовых классов, которые позволяют создавать собственные классы для реализации графического интерфейса, удобной системой общения между виджетами приложения с помощью системы сигналов и слотов и хорошей документацией, позволяющей в быстрые сроки разбираться в устройстве Qt. Данный список средств позволяет реализовать все задачи, выбранные для курсового проекта.

2.2 Моделирование программы

Архитектура программного средства является монолитной. Данный вид архитектуры легко развёртывается, простой в разработке. Отдельные части архитектуры вынесены в модули. Модули взаимодействуют через чётко определённые интерфейсы. Все части программного средства унифицированы. Основным критерием для выбора монолитной архитектуры была высокая производительность работы программного средства. Архитектура программного средства состоит из 4 основных модулей:

- модуль пользовательского интерфейса
- модуль работы с приложением
- модуль выбора алгоритма
- модуль преобразования данных

Каждый модуль отвечает за определенную задачу. Такое разделение позволяет легко поддерживать, обновлять и расширять программное средство как вертикально, так и горизонтально.

2.2.1 Модуль пользовательского интерфейса

Этот модуль предназначен для обработки запросов пользователя и передаче данных для дальнейшей обработки в модуль с основной бизнес-логикой, где происходит их дальнейшее преобразование, а также для отображения полученных данных.

Для разрабатываемого проекта создается простейший пользовательский интерфейс с помощью фреймворка Qt. Данный пользовательский интерфейс представляет собой набор пунктов меню, с которыми может взаимодействовать пользователь.

2.2.2 Модуль работы с приложением

Модуль работы с приложением необходим для взаимодействия пользователя с приложением. В данном модуле осуществляется основные операции файловой системы. Просмотр файлов и директорий нужен для ориентации пользователя и для нахождения нужных файлов. Удаление ненужных файлов и директорий является важной операцией в файловой системе. Быстрый поиск нужного файла или директории осуществляется путем ввода пути до нужного файла в строку пользователя.

2.2.3 Модуль выбора алгоритма

Данный модуль предоставляет пользователю право выбора алгоритма архивации или разархивации файлов и директорий. Так же в данном модуле реализован быстрый выбор места для архивации и разархивации данных. Выбранный алгоритм будет влиять на основную бизнес-логику программного средства.

2.2.4 Модуль архивации и разархивации

Данный модуль является основным модулем данного проекта. Модуль будет архивировать и разархивировать список файлов и директорий, которые выбрал пользователь в файл с специальном расширением.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Так как в данной работе используется объектно-ориентрованный подход программирования, то ниже приведены некоторые из классов, использованных в программе.

3.1 MainWindow

MainWindow – это главное окно графического интерфейса архиватора, с помощью которого пользователь может удобно взаимодействовать с программой.

Поля (private):
-ui:UI::MainWindow* - экземпляр главного класса (MainWindow)

-m_mapPages:QMap<int,QDialog*> - карта, которая хранит пару (ключ, значение) и обеспечивает быстрый поиск значения. Необходима генерации показа новых новых окон – “AddWindow” или “ExtractWindow”.

-model:QFileSystemModel* - доступ к локальной файловой системе.

-toolBarLineEdit:QLineEdit* - панель инструментов для показа текущего пути.

-spacer:QWidget* - фиксатор для прокрутки страницы.

-toolBarTextBeforeChanged:Qstring – строка запоминает конечный путь страницы.

Методы(private):
-init():void – метод, который инициализирует окна - “AddWindow” или “ExtractWindow”.

Методы(public):
-MainWindow(parent: QWidget* = nullptr):explicit – конструктор.

-synchronizeListFromEdit():void – синхронизация текущего пути в панели инструментов и “ListView” – отображение данных.

-createFileSystemModel():void – создаёт модель файловой системы.

-setQLineEditToTollBar():void – устанавливает текст ”QLineEdit” в панель инструментов “QToolBar” для показа текущего пути.

Методы(signals):
-emitText(text: const QString&):void – сигнал, который нужен для передачи нужного текста и соединения с окнами “AddWindow” и “ExtractWindow”.

Методы(private signals):
-on_actionAdd_files_to_archive_triggered():void – слот, который обрабатывает нажатие на кнопку добавить файл. И показывает окно “AddWindow”.

-on_actionExtract_to_triggered():void – слот, который обрабатывает нажатие на кнопку добавить файл. И показывает окно “ExtractWindow”.

-on_listView_clicked(index: const QModelIndex&) :void – слот, который обрабатывает одинарное нажатие на элемент в отображаемых данных “listView” и выводит результат в панель инструментов “QToolBar”.

-on_listView_doubleClicked(index:const QModelIndex&):void – слот, который обрабатывает двойное нажатие на элемент в отображаемых данных “listView” и открывает нужную директорию.

-upOnLevel():void – слот, который обрабатывает одинарное нажатие на кнопку “на шаг назад”.

-on_actionDelete_file_triggered():void – слот, который удаляет определённый файл.

Перечисление:
-Page{ADD_WINDOW, EXTRACT_WINDOW}:enum – константы
для генерации окон – “AddWindow” или “ExtractWindow”.

3.2 AddWindow

AddWindow – это окно графического интерфейса архиватора, с помощью которого пользователь может добавить определённый файл и выбрать способ сжатия (компрессии).

Поля (private):
-ui:Ui::Add_window* - экземпляр окна “AddWindow”.

-filesToCompress:QStringList – список строк, который содержит путь к файлу.

-direct:QString – строка, которая содержит директорию.

-pathForQLineEdit:QString – строка, которую мы получаем из списка строк.

-compressorRLE: CompressorRLE – экземпляр класса “CompressorRLE”.

-compressorHuf: CompressorHuffman – экземпляр класса “CompressorHuffman”.

Методы (private slots)
-on_BrowseButton_clicked():void – слот, который помещает путь в поле “Browse”.

-on_AppendButton_clicked():void – слот, который помещает путь в поле “Append”.

-on_Ok_clicked():void - слот, обрабатывает нажание на кнопку “Ok”.

-recvText(text:const QString&):void – слот, который соединяется с “MainWindow” для получения пути для выбранного файла.

3.3 ExtractWindow

ExtractWindow– это окно графического интерфейса архиватора, с помощью которого пользователь может распаковать (декомпрессия) ранее запакованный файл.

Поля (private):
-ui:Ui::Extract_window* - экземпляр окна “ExtractWindow”.

-filesToCompress:QStringList – список строк, который содержит путь к файлу.

-direct:QString – строка, которая содержит директорию. -pathForQLineEdit:QString – строка, которую мы получаем из списка строк.

-compressorRLE:CompressorRLE – экземпляр класса “CompressorRLE”.

-compressorHuf: CompressorHuffman – экземпляр класса “CompressorHuffman”.

Методы (private slots)
-on_BrowseButton_clicked():void – слот, который помещает путь в поле “Browse”.

-on_AppendButton_clicked():void – слот, который помещает путь в поле “Append”.

-on_Ok_clicked():void - слот, обрабатывает нажание на кнопку “Ok”.

-recvText(text:const QString&):void – слот, который соединяется с “MainWindow” для получения пути для выбранного файла.

3.4 CompressosRLE

CompressorRLE – класс, который реализовывает алгоритм сжатия RLE (Run-Length Encoding).

Методы (public):
-compressRLE(compressFileName:QString&,rleFilNam: QString&):void – метод, который реализует алгоритм сжатия файла “RLE”.

-decompressRLE(compressedRLE:QString&, decompressedFileName:QString&):void - метод, который реализует алгоритм декодирования сжатого файла “RLE”.

-IsDir(QString path):bool – метод, который проверяет является ли путь директорией или файлом.

-dirWalk(char* dirPath, QStringList fileList) – метод, который заносит все файлы и директории в массив fileList.

-getRelativePath(string compressibleFolderPath, string fullPathFile):QString – метод, который возвращает путь файла или директории относительно архивируемой папки.

3.5 CompressosHuffman

Класс CompressorHuffman – класс, который реализовывает алгоритм сжатия методом Хаффмана.

Поля (private):
-code:std::vector<bool> - создаём вектор типа bool для проверки.

-table:std::map<char, std::vector<bool>> - создаём таблицу(map) типа char и пару ключ-значение для нее типа bool (для представления в виде бинарного дерева).

-root:static Node* - создаём указатель узел.

Методы (public):
-readAllBytesFile(filename:QString&, info:std::vector<char>&):void – метод, который считывает все байты из файла.

-compressHuffman(compressFileName:QString&, hufFileName:QString&):void – метод, который будет реализоовать сжатие данных.

-decompressHuffman(hufFileName:QString&, decompressFileName:QString&):void – метод, который будет реализовывать декодирование сжатого файла (распаковка).

-buildTable(root:Node*):void – метод, который создаёт таблицу символов для кодов Хаффмана.

-Print(root:Node*, depth:int = 0):void – метод, который выводит символы в консоль.

3.6 Node

Класс Node – класс, который представляет собой “узел”, для реализации алгоритма Хаффмана.

Поля (private):
-left:Node* - левый указатель на узел. -right:Node* - правый указатель на узел.
-count:int – переменная, которая будет считать кол-во совпавших символов.

-symbol:char – переменная, которая будет хранить сам символ.

Методы (public):
-Node() – конструктор без параметров, создаёт узел без параметров т.е. пустой (новый).

-Node(symbol:char, count:int) – конструктор, который устанавливает символ и кол-во совпадений и обнуляет левый и правый узел.

-Node(left:Node*, right:Node*) – конструктор, который создаёт узел где-то в середине и поэтому уже известны его ветви левая и правая.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

В данном разделе будет рассматриваться реализация функционала, который используется программой для работы, а также алгоритм работы программы.

Для сжатия данных придумано множество техник. Большинство из них комбинируют несколько принципов сжатия для создания полноценного алгоритма. Даже хорошие принципы, будучи скомбинированы вместе, дают лучший результат.

Кодирование длин серии (RLE).

Это алгоритм заменяет серии из двух или более одинаковых символов числом, обозначающим длину серии, за которым идёт сам символ. Полезен для сильно избыточных данных, типа картинок с большим количеством одинаковых пикселей, или в комбинации с алгоритмами типа BWT.[3][4]

Пример:
На входе: AAABBCCCCDEEEEEEAAAAAAAAAAAAAAAAAAAA
На выходе: 3A2B4C1D6E21A

Алгоритм Хаффмана

Идея алгоритма: зная вероятность вхождения символов в сообщение, можно описать процедуру построения кодов переменной длинны состоящих из целого количества битов. Символами с большей вероятностью присваиваются более короткие коды.

Динамический алгоритм Хаффмана на входе получает таблицу частот встречаемости символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмана.[5][6]

Рассмотрим алгоритм по шагам:

1. Символы входного алфавита образуют список свободных узлов. Каждый узел имеет кол-во вхождений символа в ожидаемое сообщение, сам символ, и два указателя – на левый и правый узел.
2. Для удобства можно отсортировать узлы (сортировать можно по кол-ву вхождений или по ascii-кодам символов).
3. Выбираются 2 наименьших свободных узла дерева.
4. Создается родительский узел, имеющий указатели на созданные 2 узла (левый и правый).
5. Родитель добавляется в список свободных узлов, а двое его потомков удаляются из этого списка.

6. Одной дуге выходящей их родителя ставится в соответствие бит 1, другой – бит 0.
7. Далее пункты повторяются, начиная со второго, до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева.

4.1 Архивация файлов и директорий алгоритмом RLE compressRLE

Данный метод реализует архивацию данных с помощью алгоритма RLE. Метод принимает на вход строку с путем выбранного файла или директории, а также строку с путем архива. Далее она формирует массив относительных путей вложенных файлов и директорий с помощью dirwalk, который будет описан далее, и переходит к циклу записи их в архив.

```
for(const auto &it: listFile) {
    auto relativePath = getRelativePath(compressFileName.toStdString(), it.toStdString());
        fileCompressed << relativePath.toStdString();
        if(IsDir(it)) { fileCompressed << " $"; }
        fileCompressed << '\n';
}
```

После записи путей файлов и директорий идет цикл кодировки данных файла с помощью алгоритма RLE и запись их в архив.

4.2 Разархивация архива алгоритмом RLE decompressRLE

Данный метод реализует разархивацию данных с помощью алгоритма RLE.

Метод принимает на вход строку с путем архива и строку с путем директории, куда будет проходить разархивация. Метод циклом считывает относительные пути файлов и директорий из архива и заносит их в очередь.

```
while(ch1 != '\n' || dirToDescompress.peek() != '\n') {
        QString filePath;
        dirToDescompress.get(ch1);
        while(ch1 != '\n') {
           filePath.append(ch1);
           dirToDescompress.get(ch1);
        }
        filePath.prepend(decompressedDirPath);
        queueFilesPath.enqueue(filePath);
}
```

Очередь была выбрана для того, чтобы не было ошибки при создании файлов и директорий. После идет цикл чтения закодированных данных в архиве и запись их в файл, который взят из очереди.

4.3 Нахождение всех вложенных файлов и директорий dirWalk

Данный метод заполняет массив путями всех вложенных файлов и директорий. На вход принимает путь выбранной папки и массив строк, который будет заполняться путями. Метод сначала открывает поток выбранного каталога и возвращает указатель на структуру типа DIR, которая содержит информацию о каталоге. Потом идет цикл readdir, с помощью которого обрабатывается каждый файл и директория. Если в директории есть файл, то его путь заносится в массив строк путей. Данный метод рекурсивный, если в выбранной директории есть вложенная директория.

4.4 Получение относительного пути getRelativePath

Данный метод возвращает относительный путь файла или директории относительно архивированной директории. На вход метод принимает строку с путем архивированной директории и строку с путем самого файла или директории. Далее метод находит позицию пути архивированной в пути самого файла и циклом удаляет путь архивированной директории.

```
QString CompressorRLE::getRelativePath(string compressibleFolderPath, string fullPathFile) {
        string::size_type pos = fullPathFile.find(compressibleFolderPath);
          while (pos != string::npos) {
             fullPathFile.erase(pos, compressibleFolderPath.size());
             pos = fullPathFile.find(compressibleFolderPath, pos + 1);
          }
          return QString::fromStdString(fullPathFile);
    }
```

В итоге мы получаем алгоритм работы программы:

1. Начало программы. Взаимодействие пользователя с графическим интерфейсом
2. Получение файла или директории для архивации
3. Определение алгоритма
4. Реализация того или иного алгоритма
5. Создание заархивированного файла.
   5.1. При необходимости получения файла для разархивации
   5.2. Выбор соответствующего алгоритма декомпрессии, в зависимости от файла
   5.3. Создание разархивированного файла.

5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Тестирование — это наблюдение за работой приложения в разных искусственно созданных ситуациях. Данные, полученные в ходе тестирования, важны при планировании последующей стратегии развития приложения. Это своего рода диагностика, которая влияет на многие дальнейшие действия.

Для тестирования разработанного приложения были сымитированы ошибки, чтобы проверить, как приложение на них реагирует.

В данном приложении реализован удобный пользовательский интерфейс. На практике не всегда пользователь может выбрать правильную директорию или файл, из-за чего приложение не будет выполнять выбранную операцию. Пользователь может не знать причину остановки приложения. Для решения данной проблемы реализованы помогающие окна с возникшей ошибкой.

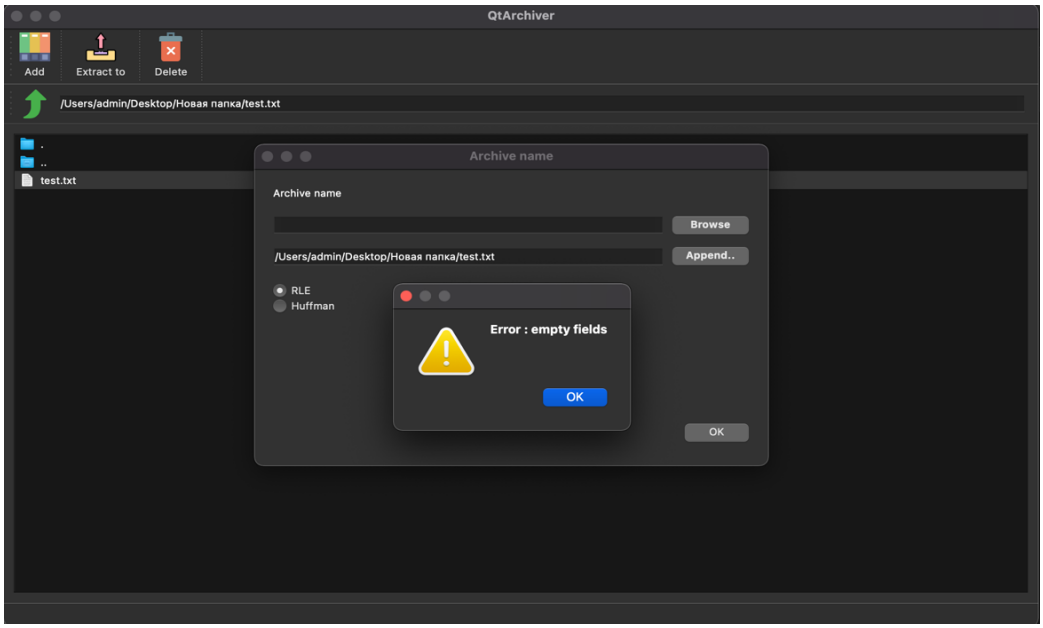

Рисунок 5.1 — Ошибка при выборе директории.

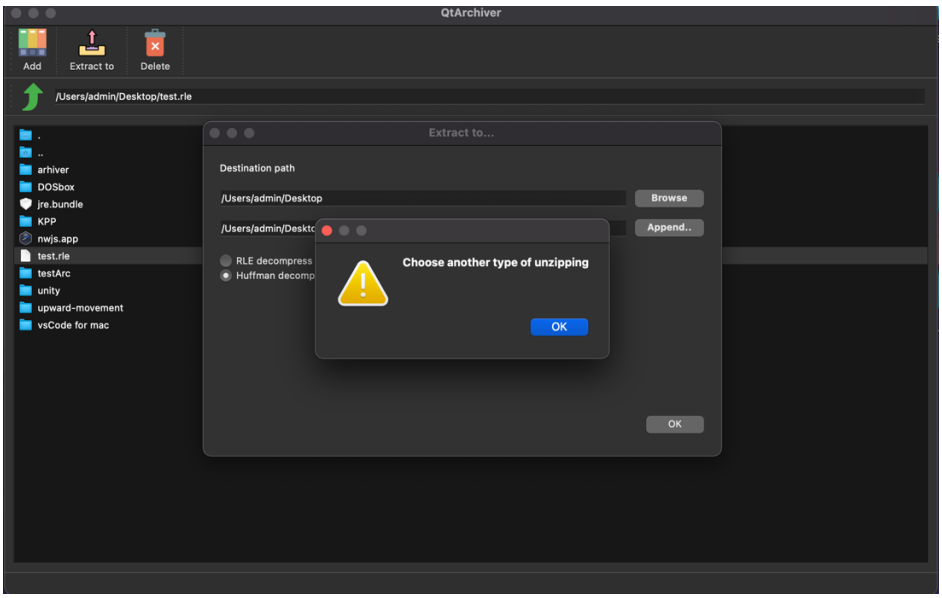

Рисунок 5.2 — Ошибка при выборе алгоритма разархивации.

Разархивация файлов происходит только с расширениями .rle и .huf.

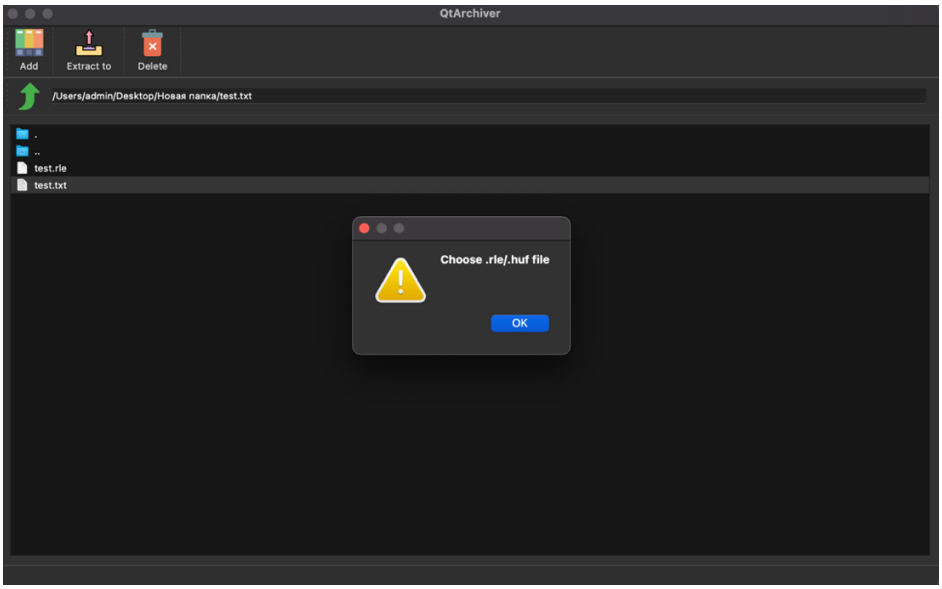

Рисунок 5.3 — Ошибка при выборе разархивации файла.

В данном разделе были описаны все ключевые ошибки, которые могут возникнуть при использовании разработанного приложения. Было проведено тестирование и проработаны возможные решения данных моментов. После проверки на возникновение ошибок, приложение сработает корректно и не вызовет никаких нарушений в работе устройства.

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

После запуска программного средства пользователь попадает на основное окно. Окно представляет список файлов и директорий выбранной папки, путь которой можно ввести(Рисунок 6.1).

Рисунок 6.1 — Окно программы при запуске.

Пользователю сразу предоставляется выбор действий с программой. Также пользователь сразу видит полный путь до текущей директории(Рисунок 6.2).

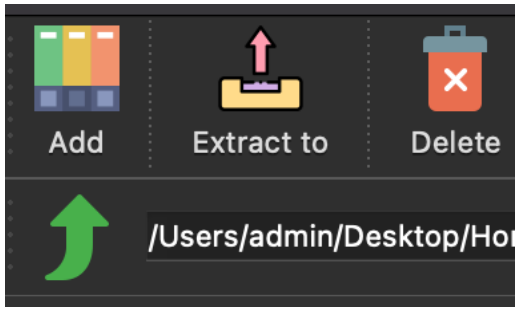

Рисунок 6.2 — Меню выбора операций.

Команда Add архивирует выбранный файл или директорию (который можно выбрать в списке файлов или в всплывающем окне) в выбранную директорию(Рисунок 6.3).

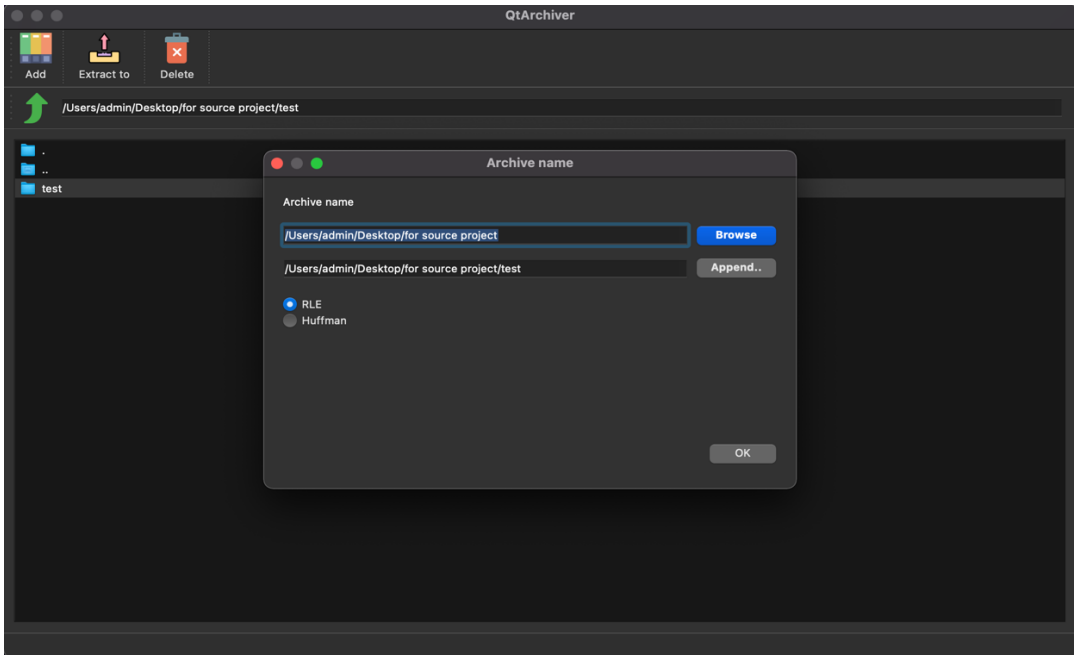

Рисунок 6.3 — Меню команды Add.

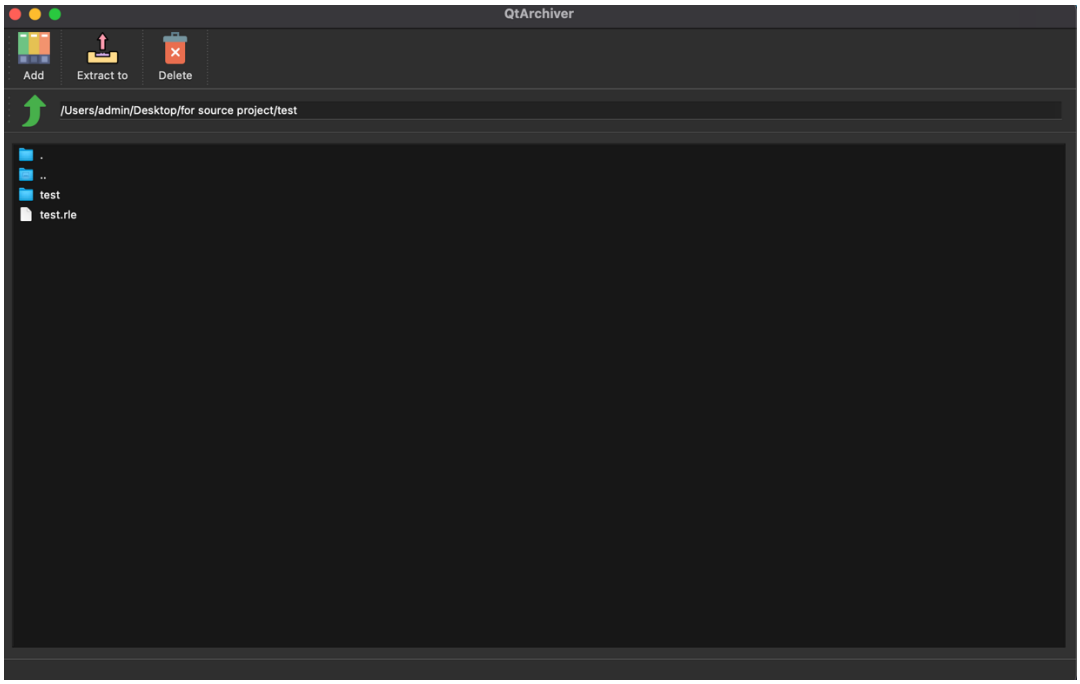

Рисунок 6.4 — Результат операции Add.

Команда Extract to разархивирует файл (который можно выбрать в списке файлов или в всплывающем окне) формата .rle и .huf в выбранную директорию(Рисунок 6.5 и Рисунок 6.6).

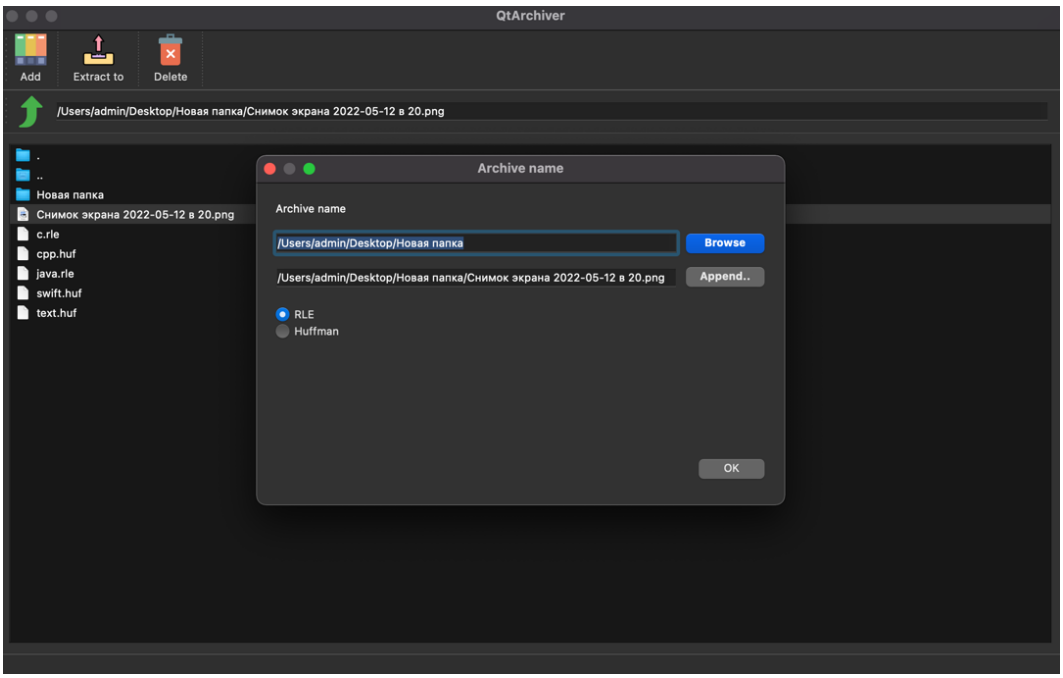

Рисунок 6.5 — Меню команды Extract to.

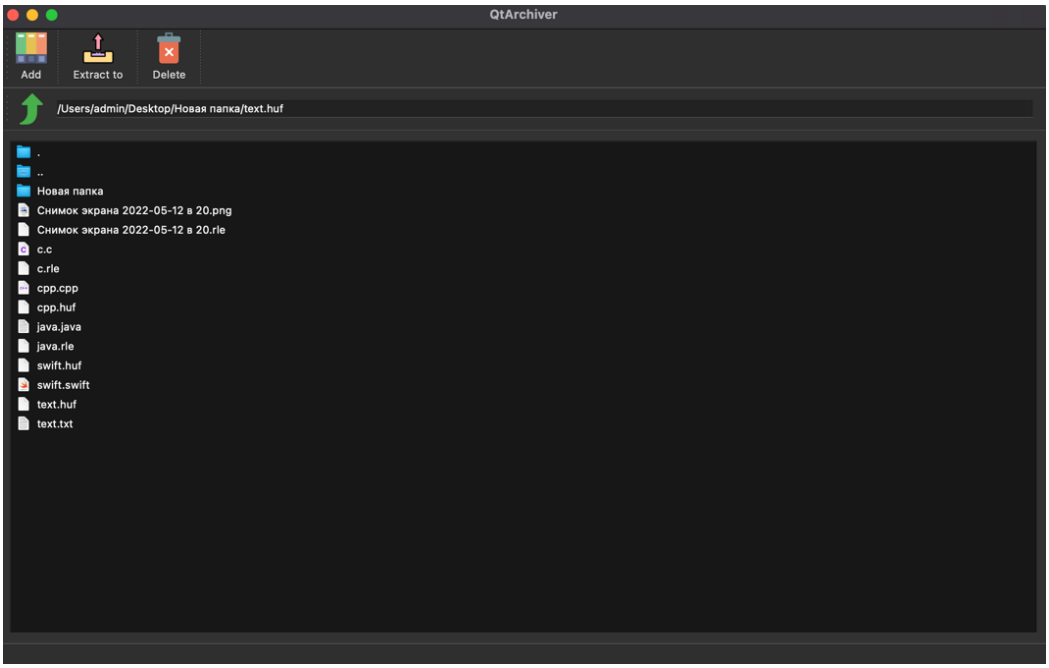

Рисунок 6.6 — Результат команды extract to.

Команда Delete удаляет выбранный файл или директорию из списка. Команда Delete не переносит файл в корзину, а удаляет его полностью из файловой системы(Рисунок 6.7 и Рисунок 6.8).

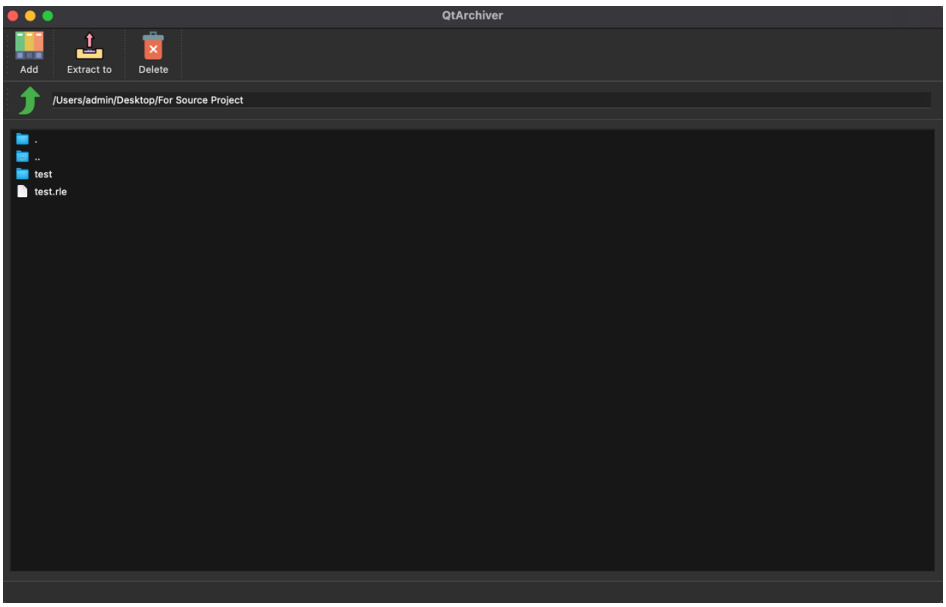

Рисунок 6.7 — Директория до удаления файла.

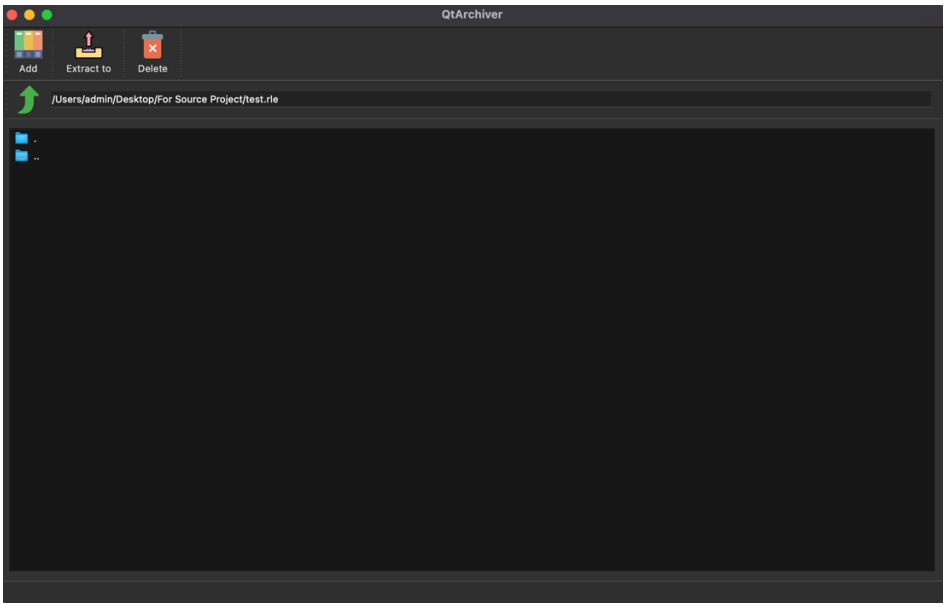

Рисунок 6.8— Результат после операции Delete

ЗАКЛЮЧЕНИЕ

В данном разделе будут проведены итоги разработки программного обеспечения.

В результате выполнения курсового проекта были изучены темы проектирования приложения, алгоритмы архивации и разархивации файлов и были углублены знания языка программирования C++ и фреймворка Qt и с графическим интерфейсом.

Работа была разделена на такие этапы, как анализ существующих аналогов, литературных источников, постановка требований к проектируемому программному продукту, системное и функциональное проектирование, конструирование программного продукта, разработка программных модулей и тестирование проекта. После последовательного выполнения вышеперечисленных этапов разработки было получено исправно работающее приложение.

В дальнейшем будет доработан пользовательский интерфейс, функционал приложения, а также добавлены новые алгоритмы архивации и другие типы файлов.

ЛИТЕРАТУРА

[1]. Лафоре, Р. Объектно-ориентированное программирование с C++ /4-е издание

[2]. Лав Р. Системное программирование на Linux/ 2-е издание 2014.

[3]. Документация по сжатию по алгоритму RLE – [Электронный ресурс]. – Адрес ресурса: https://www.cplusplus.com/forum/beginner/28542/

[4]. Документация по сжатию по алгоритму RLE – [Электронный ресурс]. – Адрес ресурса: https://habr.com/ru/post/141827/

[5]. Документация по сжатию по алгоритму Хаффмана – [Электронный ресурс]. – Адрес ресурса: http://algolist.ru/compress/standard/huffman.php

[6]. Документация по сжатию по алгоритму Хаффмана – [Электронный ресурс]. – Адрес ресурса: https://habr.com/ru/company/otus/blog/497566/

[7]. Документация по формату Rar – [Электронный ресурс]. – https://filesreview.com/ru/info/rar

[8]. Документация по формату Zip – [Электронный ресурс]. – https://experience.dropbox.com/ru-ru/resources/what-is-a-zip-file

[9]. Архиватор – понятие и назначение. – [Электронный ресурс]. – https://myblaze.ru/chto-takoe-arhivator-i-zachem-on-nuzhen-printsip-rabotyi-arhivatora/

[10] Сжатие данных – понятие и назначение. – [Электронный ресурс]. – https://habr.com/ru/post/231177/

[11] Классификация методов сжатия. – [Электронный ресурс]. – https://mf.grsu.by/UchProc/livak/po/comprsite/theory_classification_01.html

ПРИЛОЖЕНИЕ А

(обязательное)

Структурная схема приложения

ПРИЛОЖЕНИЕ Б

(обязательное)

Диаграмма классов

ПРИЛОЖЕНИЕ В

(обязательное)

Ведомость документов