ВВЕДЕНИЕ

Иногда приходится переместить некоторую информацию из одного места в другое. Однако некоторые файлы занимают слишком много места, и их пересылка через ограниченное по скорости сетевое подключение занимет достаточно много времени. Также, место назначения, куда нужно скопировать файл, может быть ограничено в объеме. В таких случаях просто необходимо как-то сократить объем файла. Для этой задачи пользователи ПК используют архиваторы файлов и директорий. Все архиваторы делятся на те, которые применяют алгоритм сжатия с потерями, и те, что сжимают без потерь. Хоть на первый взгляд и кажется, что сжатие с потерями не имеет смысла, ведь, сжав, например, текстовый файл с каким-то текстом, разархивировав его мы не сможем получить осмысленный текст. Однако эти программы активно применяются в области сжатия аудиозаписей, где некоторая потеря информации не сильно сказывается на восстановленном файле.

Степень сжатия данных у алгоритма без потерь ниже, чем у сжатия с потерями. Однако в курсовом проекте используется именно алгоритм сжатия без потерь по алгоритму Хаффмана, что позволит сжимать все виды файлов и вновь использовать их после восстановления. По этому алгоритму каждый символ в файле заменяется его битовой кодировкой, которая создается с помощью построения сбалансированного бинарного дерева, где в листьях хранятся символы и количество их повторений в файле, а в узлах хранятся числа, которые равны сумме чисел в дочерних узлах.

В конечном итоге программа позволяет сжимать файлы и директории, создавая архив. Также программа способна провести разархивацию сжатой информации, восстановив исходные данные без потерь. Для того, чтобы восстановить данные, необходимо как-то восстановить битовые коды символов. Поэтому в начале архива помещается информация для программы, в которую записывается информация о необходимых папках, а перед сжатыми записями файлов находятся словари кодировок, где к каждому встреченному в исходном файле байту сопоставляется новый битовый код. Самая высшая степень сжатия по данному алгоритму у текстовых файлов. Это обусловлено тем, что в текстовых файлах вы редко встретите более 130 различных видов байт, что позволяет построить относительно небольшое дерево. С помощью дерева каждый байт, состоящий из 8 бит, может быть заменен другим битовым кодом. К примеру, самым часто встречающимся символом в текстовых файлах является пробел. Однако он занимает столько же места, сколько и любой другой символ. При построении дерева учитывается число повторений символа в файле, таким образом символ пробела окажется в самой вершине дерева. Особенность бинарного дерева в том, что путь к каждому листу дерева уникален. Если взять проход влево за единицу, а проход вправо за ноль, то путь от вершины к каждому узлу сгенерирует свой уникальных бинарный код. Поэтому путь к редко встречающимся символам будет несколько длиннее восьми бит, обычно около 12, однако в это же время путь к пробелу может

состоять всего из двух бит. А учитывая то, что пробелы встречаются намного чаще, чем те символы, код которых становится больше восьми бит, получается сжать информацию.

В общем случае архиватор хорошо показывает себя и с другими типами информации. Так некоторые изображения получается сжать в два раза. Но основной проблемой является то, что аудиозаписи, видеофайлы и многие другие форматы информации содержат в себе все 256 различных возможных байт, из-за этого не всегда получается уменьшить размер файла.

Работа программы сосредоточена в трех потоках: потоке архивации, потоке разархивации и основном потоке приложения, где обрабатывается графический интерфейс, а также ввод пользователя и вывод информации для него.

Программ, способных на архивацию данных достаточно много, но в большинстве своем это довольно старое ПО с малоудобным интерфейсом. Поэтому основной целью данного курсового проекта является создание не только быстрого, легкого, но и удобного для пользователя приложения по архивации файлов и директорий без потерь информации.

Для разработки данного курсового проекта был выбран известный фреймворк Qt. Сам по себе он очень удобен для разработчика, так как содержит в себя обширную библиотеку классов. Также Qt нацелен на создание пользовательского интерфейса. Языком написания проекта является C++, он хорошо знаком с предыдущего года обучения, а также довольно быстр, по сравнению с другими языками программирования, что позволит ускорить работу архивации и разархивации данных.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Обзор Аналогов

Обычно архиваторы используют разные алгоритмы сжатия данных, детали которых разработчики скрывают друг от друга. Следовательно, порождается множество различных форматов, каждый из которых требует свою программы для работы с содержимым архива. Среди множества возможных аналогов данного курсового проекта, наиболее выделяются 2:

1. WinRAR это архиватор RAR для Windows — мощный инструмент для архивирования и управления архивами. Интерфейс программы представлен на рисунке 1.1.1. Помимо Windows существуют версии RAR для других операционных систем — Linux, FreeBSD, macOS, Android. WinRAR полностью поддерживает типы RAR и ZIP 2.0, а также способен работать с форматами 7Z, ARJ, BZ2, CAB, GZ, ISO, JAR, LZ, LZH, TAR, UUE, XZ, Z, 001 и ZIPX нескольких типов. Особенностью WinRAR является использование собственного высокоэффективного алгоритма сжатия данных с поддержкой непрерывного (solid) архивирования. Несмотря на лицензирование своего ПО, разработчик предоставляет доступ к исходному коду разархиватора, тем самым предоставляя другим разработчикам возможность разархивации файлов формата RAR.

2. 7ZIP – бесплатное программное обеспечение с открытым исходным кодом. Для работы использует расширение 7z, но также полностью поддерживает форматы XZ, BZIP2, GZIP, TAR, ZIP и WIM. Частично поддерживает работу с форматами AR, ARJ, CAB, CHM, CPIO, CramFS, DMG, EXT, FAT, GPT, HFS, IHEX, ISO, LZH, LZMA, MBR, MSI, NSIS, NTFS, QCOW2, RAR, RPM, SquashFS, UDF, UEFI, VDI, VHD, VMDK, WIM, XAR и Z. Из преимуществ, архиватор 7ZIP показывает наибольшую эффективность в работе с ZIP и GZIP форматами среди аналогов. Интерфейс программы представлен на рисунке 1.1.2.

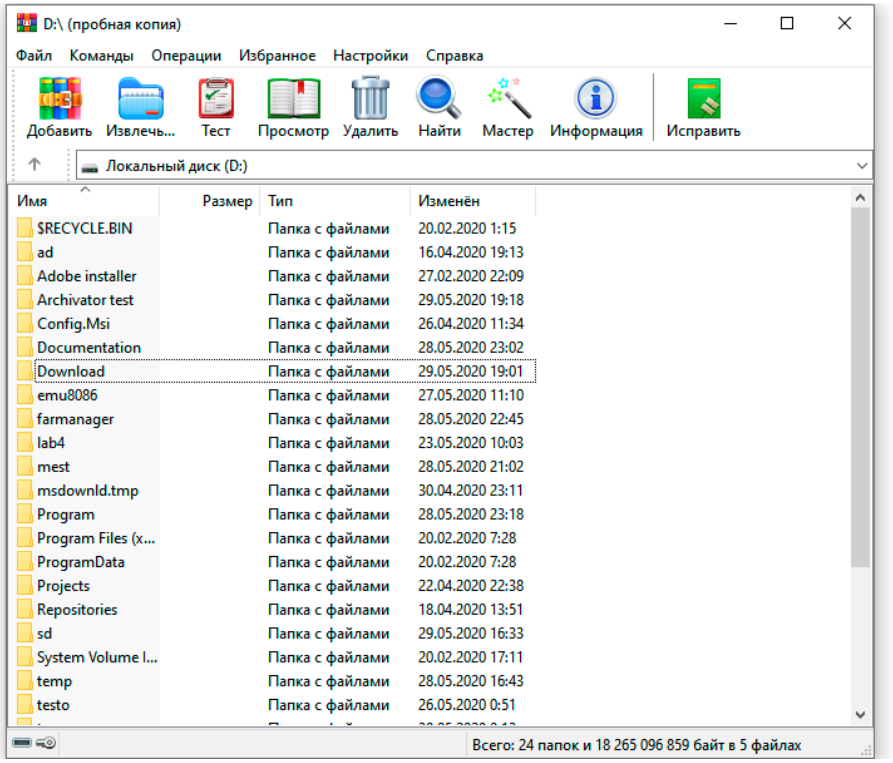

Рисунок 1.1.1

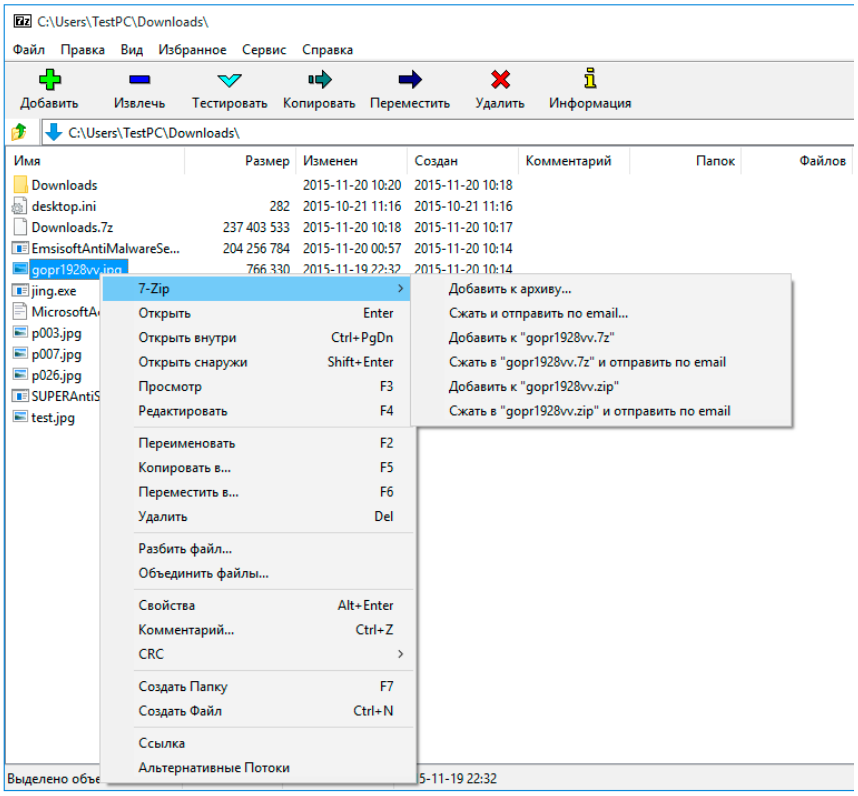

Рисунок 1.1.2

Преимуществами предоставленных аналогов данного курсового проекта являются их надежность, они были проверены временем, и высокая эффективность алгоритмами сжатия, но для простого пользователя эти

программы имеют множество непонятных функций, которыми большинство простых пользователей так никогда и не воспользуется. Данный курсовой проект имеет простой, но удобный пользовательский интерфейс, который предоставляет пользователю то, что ему необходимо – архивацию и разархивацию данных.

1.2 Обзор средств разработки

Знакомство с алгоритмом Хаффмана и его реализацией на языке С можно найти на веб-ресурсе [1]. Тут в нескольких статьях описывается создание программы по архивации данных. Также можно найти сравнение быстродействие данного алгоритма на различных языках.

В книге [5] можно познакомится с языком С++, а также в ней рассматривается разработка различных полезных алгоритмов.

Книга [4] позволяет глубже взглянуть в разработку приложений на языке С++, знакомит с многими неочевидными на первый взгляд особенностями языка, постепенно усложняясь к концу книги и давая все более точное представление о языке.

Информационный портал [3] выкладывает переводы зарубежных уроков по языку С++. Причем информация тут доносится на простом языке с множеством различных примеров в коде, что позволяет легко разобраться в новой для читателя теме.

Основная информация по разработке приложения с помощью фреймворка Qt предоставляется на официальном сайте документации фреймворка [6]. Тут находится подробное описание классов и методов, приведены примеры кода и рисунки, отражающие конечный результат реализации той или иной возможности фреймворка. Те проблемы, что не получается решить с помощью документации Qt, помогает решить форум [2]. Большинство вопросов, которые возникают у разработчика, так или иначе могли появиться и у других разработчиков, которые те задали на форуме и получили детальный ответ от других пользователей. Если же на ваш ответ не нашлось ответа, вы можете задать свой, и отзывчивое сообщество форума в скором времени постарается помочь вам.

2 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ

При разработке любой программы всегда стоит группировать отдельные ее элементы в функциональные блоки, что упростит понимание программы, а также повысит ее эффективность. Так как данное приложение разрабатывается в среде разработки Qt, то возможно использование стандартная система управления приложением, написанном в данном фреймворке, основанное на слотах и сигналах. Традиционно такое приложение можно поделить на два блока: блок управления приложением и блок пользовательского интерфейса.

Структурная схема предоставлена в Приложении А.

2.1 Модуль управления приложением

Данный модуль сосредотачивает в себе всю основную логику приложения - в данном случае работа с файлами и алгоритм архивации/разархивации. Этот модуль делится на две части: набор классов для создания архива и набор классов для распаковки архивов. Каждый из этих модулей работает в своем потоке, что позволяет модулю интерфейса продолжать свою работу независимо от того, происходит в данный момент операции архивации или разархивации, или нет. Во время работы оба потока обрабатывают свое состояние, проверяя, требуется ли от них прерваться или нет. У модуля пользовательского интерфейса имеется возможность прервать работу потока, для этого ему нужно уведомить поток. В свою очередь, поток, увидев этот сигнал, моментально завершит операцию.

Этот модуль содержит в себе:

1) Блок пользовательского интерфейса

2.2 Модуль пользовательского интерфейса

В данном модуле осуществляется вывод необходимой для пользователя информации на экран, а также получение входных данных. Для удобного выбора файла используется отображение файловой системы в древовидной форме. Снизу приложения отображаются кнопки взаимодействия с файлами и директорий. Взаимодействие приложения с пользователем осуществляется с помощью диалоговых окон. Весь интерфейс приложения работает на системе слотов и сигналов Qt (рисунок 2.2.1). Слот одного объекта связывается с сигналом другого. Сигнал – метод без тела, может быть с аргументом, слот – метод с телом. При испускании сигнале в одном объекте произойдет вызов связанного с ним метода в другом. Так, например, в данной программе связано событие конца ввода имени файла пользователя и переход к методу начала архивации.

Этот модуль содержит в себе:

1) Блок обработки входных данных для архивации.
2) Блок обработки содержимого файлов.

3) Блок архивации на основе обработанных данных.
4) Блок обработки содержимого архива.
5) Блок разархивации по обработанным данным.

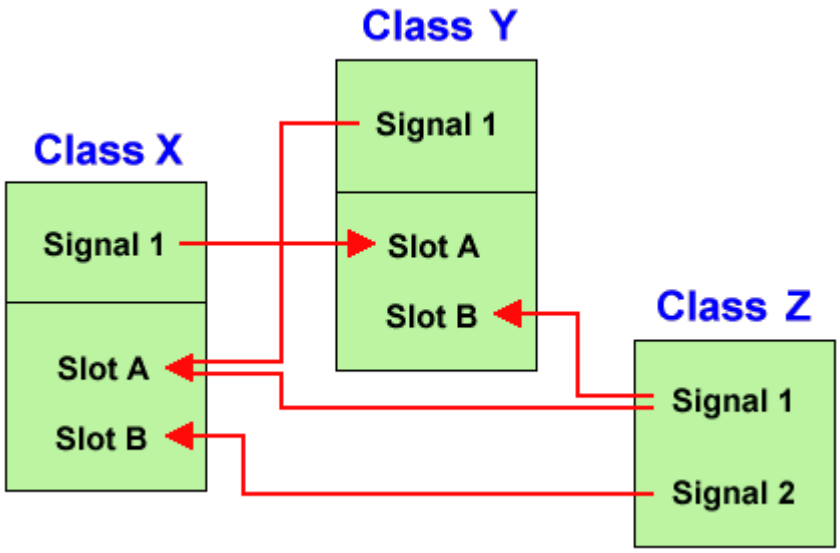

Рисунок 2.2.1

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В этом разделе описывается функционирование разрабатываемого программного обеспечения, классы и их методы.

Диаграмма классов предоставлена в Приложении Б.

3.1 Описание функционирования программы

Работу данного программного обеспечения можно описать следующей цепочкой действий:

1) Выбор файла с помощью удобного отображения файлов в древовидной структуре, что обеспечивается блоком пользовательского интерфейса.

2) Выбор действия (архивация, разархивация или удаление).

3) Если для архивации была выбрана директория, а не файл, то первым делом происходит поиск всех вложенных в директорию файлов. Это происходит в блоке обработки входных данных для архивации.

4) Обработка содержимого файла (составление бинарного дерева для архивации и чтение таблицы кодировок для разархивации), данные действия обрабатываются в блоке обработки содержимого файлов и блоке обработки содержимого архива соответственно.

5) Запись в новый файл, причем Архиватор читает исходный файл небольшими порциями (к примеру, по 256 байт), каждый байт заменяется кодировкой, полученной путем составления бинарного дерева из всех уникальных байтов исходного. Полученный в ходе этого процесса буфер сжатой информации записывается в выходной файл. Так программа продолжает считывать новые порции исходных данных до тех пор, пока весь исходный файл не будет переведен в свою сжатую форму (рисунок 3.1), это происходит в блоке архивации на основе обработанных данных. Разархиватор начинает с чтения дерева каталогов из архива и создания папок, далее все файлы архива по-одному разархивируются в соответствующие папки на основе кодировок байтов, которые записаны в архиве перед каждой сжатой записью (рисунок 3.2), это обеспечивается блоком разархивации файла на основе обработанных данных.

7) Если Пользователь прервал операцию архивации или разархивации, то программа перейдет к соответствующему слоту обработки этого сигнала, так при прерывании операции архивации файл архива будет удален.

8) Далее программа переходит к блоку пользовательского интерфейса.

Рисунок 3.1.1 – Логика архивации файла

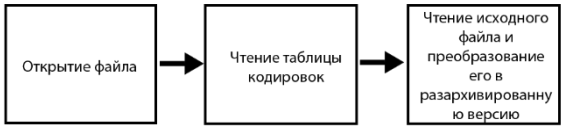

Рисунок 3.1.2 – Логика разархивации файла

3.2 Описание структуры взаимодействия внутри приложения

3.2.1 Шаблонный класс Node

Данный класс является узлом структуры данных двоичного дерева поиска (класс `bTree`), является шаблоном двух типов: T и N, где T – тип, значение которого будет использовано для сортировки дерева, а N – тип поля, которое будет иметь значение только в листах дерева (лист не имеет наследников и в нем определено значение поля `value`).

Таблица 3.2.1 – Шаблонный класс Node

Private поля класса		
Имя	Тип	Описание
`has_value`	`bool`	Необходимо для различия между листом и узлом дерева
Public поля класса		
Имя	Тип	Описание
1	2	3
`code`	`QString`	Содержит кодировку элемента в листе
`count`	`T`	В листе содержит число повторений байта в файле, а в узле является суммой значений этого поля в наследниках

Продолжение таблицы 3.2.1

1	2	3
value	N	Значение узла дерева
left	Node<T, N>*	Указатель на левого наследника узла
right	Node<T, N>*	Указатель на правого наследника
endNode	bool	Если эта переменная истинна, то данный узел создан с целью получения уникального кода, который используется как маркер конца сжатой записи в файле

Public методы класса			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
1	2	3	4
Node	T cnt	-	Конструктор узла с одним параметром – значением поля count
Node	T cnt, N val	-	Конструктор нового узла, устанавливает значение поля has_value true

Продолжение таблицы 3.2.1

1	2	3	4
Node	Node<T, N>* lef, Node<T, N>* rgt	-	Конструктор нового узла на основе двух существующих. Особенность в том, что has_value = false, а значение поля count вычисляется как сумма аналогичных полей обоих наследников
~Node	-	-	Вызывает delete для наследников
getCount	-	T	Возвращает значение поля count
getValue	-	N	Возвращает значение поля value
getCode	-	QString	Возвращает битовую кодировку листа
hasValue	-	bool	Возвращает значение поля has_value
setEndNode	bool t	void	Устанавливает текущий узел как узел с кодом маркировки конца архива
isEndNode	-	bool	Возвращает значение поля endNode
operator>	Node<T, N> other	bool	Сравнивают два узла по значению поля count
operator<			

Продолжение таблицы 3.2.1

1	2	3	4
operator==	Node<T, N> other	bool	Сравнивают два узла по значению поля count
operator!=			

3.2.2 Класс NodeComparator

Класс-компаратор, предназначен для упорядочивания узлов по возрастанию значения поля count в различных контейнерах, в частности в очереди с приоритетом.

Таблица 3.2.2 – Класс NodeComparator

Public методы класса			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
operator()	Node<int, char> *first, Node<int, char> *second	bool	Компаратор, который используется в приоритетной очереди, для создания приоритета по возрастанию значения

3.2.3 Класс bTree

Структура данных Бинарное дерево поиска, необходимо в алгоритме Хаффмана для составления битовых кодировок узлов.

Таблица 3.2.3 – Класс bTree

Private поля класса		
Имя	Тип	Описание
1	2	3
root	Node<int, char> *	Главный узел дерева

Продолжение таблицы 3.2.3

1	2		3
dictionary	QMap<char, QString>		Словарь кодировок
endCode	QString		Двоичный код маркера конца архива
Public методы класса			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
bTree	Node<int, char> *nd	bTree	Конструктор дерева из существующего узла
~bTree	-	-	Вызывает рекурсивный метод destroyTree
destroyTree	Node<int, char> *&node	-	Рекурсивно удаляет все узлы дерева
formCodes	-	-	Вызывает метод formCodesRec
formCodesRec	Node<int, char>* node, QString tempCode	-	Проходит по листьям дерева и составляет для них коды, которые записываются в поле dictionary
getDictionary	-	QMap<char, QString> dictionary	Возвращает заполненный словарь – поле dictionary
getEndCode	-	QString	Возвращает строку двоичного кода маркера конца

3.2.4 Класс Catalog

Данный класс предназначен для подсчета числа повторений каждого уникально байта в входном файле.

Таблица 3.2.4 – Класс Catalog

Private поля класса			
Имя	Тип		Описание
`catalog`	`QMap<char, int>`		Словарь, где хранится байт и число его повторений в файле
Public методы класса			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
`Catalog`	`QByteArray info`	-	Конструктор, принимающий информацию, которой класс дополнит свое поле `catalog`
`add`	`QByteArray info`	`QMap<char, int>&`	Пересчитывает переданные байты
`getCatalog`	-	`QMap<char, int>&`	Возвращает словарь `catalog`

3.2.5 Класс Coder

Класс Coder предназначен для перевода входной информации в сжатую форму при помощи словаря кодировок, полученного с помощью бинарного дерева поиска `bTree`.

Таблица 3.2.5 – Класс Coder

Private поля класса			
Имя	Тип		Описание
endCode	QString		Строка с бинарным кодом маркера конца архива
prev	char		Незаполненный байт с прошлого вызова метода кодировки буфера
leftPrev	bool		истина, если prev не пуст
prevSize	short		Число бит в prev
len	int		Длина полученного из словаря кода
dictionary	QMap<char, QString>		Словарь кодировок
Public методы класса			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
1	2	3	4
Coder	-	-	Конструктор кодировщика, инициализирует поля leftPrev и prevSize
hasPrev	-	bool	Возвращает значение поля leftPrev
getPrev	-	char	Возвращает значение поля prev
setDictionary	QMap<char, QString>& dict	-	Устанавливает значение поля dictionary

Продолжение таблицы 3.2.5

1	2	3	4
getDictionary	-	QMap<char, QString> dict	Возвращает значение поля dictionary
setEndCode	QString code	-	Устанавливает значение поля endCode
getEndCode	-	QString	Возвращает значение поля endCode
getEof	-	QByteArray	Возвращает битовый код, содержащий незаконченный байт prev и endCode
encode	char sb	QByteArray	Возвращает закодированный по словарю байт
getNextCodeBuffer	QByteArray source	QByteArray	Возвращает сжатую информацию из source
clear	-	-	Очищает все поля класса Coder
static Public методы			
formByte	QString code	char	Переводит кодировку из строки из 8 нулей и единиц в байты
formBytes	QString code	QByteArray	Переводит кодировку из строки любой длины в байты

3.2.6 Класс DialogWindow

Класс предназначен для диалога с пользователем. Окно запрашивает имя файла. Кнопка ok становится доступна только при вводе хотя бы одного символа. Это реализуется с помощью системы слотов и сигналов. При

завершении вода, объект окна посылает сигнал о завершении операции, который поступает в соответствующий ему слот класса `MainWindow`.

Таблица 3.2.6 – Класс DialogWindow

Private поля класса		
Имя	Тип	Описание
`message`	`QLabel *`	Информация в окне
`input`	`QLineEdit *`	Строка ввода
`ok`	`QPushButton *`	Кнопка окончания ввода
`cancel`	`QPushButton *`	Кнопка прерывания ввода

Public методы класса			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
`DialogWindow`	`QWidget *parent = nullptr`	-	Конструктор диалогового окна

Private слоты			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
`on_text_changed`	`QString str`	-	Слот для сигнала изменения текста в строке ввода
`on_ok_button_clicked`	-	-	Слот для сигнала нажатия кнопки `ok`
`on_cancel_button_clicked`	-	-	Слот для сигнала нажатия на кнопку отмены

Продолжение таблицы 3.2.6

Сигналы			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
`fileNameEntered`	`QString str`	-	Сигнал конца ввода

3.2.7 Класс FileCollector

Класс для сбора всех файлов в директории, а также их абсолютных и относительных путей.

Таблица 3.2.7 – Класс FileCollector

Private поля класса			
Имя	Тип		Описание
`startDir`	`QDir`		Папка, с которой начнется сбор файлов
`allFiles`	`QList<FileEntry>`		Список информации о файлах в директории
`allDirs`	`QStringList`		Список относительных путей папок
Public методы класса			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
`FileCollector`	`QDir dir`	-	Конструктор сборщика файлов
`collectFiles`	-	`QList<FileEntry>`	Функция обертка
`collect_files`	`QDir current, QString relativePath`	-	Рекурсивно собирает все файлы в папке

Продолжение таблицы 3.2.7

Static public методы			
dirSize	QString dirPath, int size, const int max	int	Вернет число меньше max, если папка достаточно неглубокая, и max+1, если папка слишком глубокая
isCorrectFileName	QString path	bool	Проверяет переданный путь на содержание запрещенных символов, а также на некорректные имена

3.2.8 Класс FileEntry

Класс FileEntry содержит полный путь к файлу, относительный путь к файлу и имя файла, используется в классе FileCollector.

Таблица 3.2.8 – Класс FileEntry

Private поля класса		
Имя	Тип	Описание
path	QString	Путь к файлу
relativePath	QString	относительный путь к файлу
fileName	QString	Имя файла

Продолжение таблицы 3.2.8

Public методы класса			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
FileEntry	QString path_, QString relativePath_, QString filename_	-	Конструктор файловой записи, устанавливает все поля класса
setPath	QString path_	-	Устанавливает значение поля path
getPath	-	QString	Возвращает значение поля path
setRelativePath	QString relativePath_	-	Устанавливает значение поля relativePath
getRelativePath	-	QString	Возвращает значение поля relativePath
setFileName	QString fileName_	-	Устанавливает значение поля fileName
getFileName	-	QString	Возвращает значение поля fileName

3.2.9 Класс FileDecoder

Класс для разархивации сжатой записи. Сначала он считывает и создает нужные папки, затем по-одному считывает и распаковывает файлы, создавая их в нужных папках.

Таблица 3.2.9 – Класс FileDecoder

Private поля класса			
Имя	Тип		Описание
`input`	`ReadBuffer`		Буфер считывания архива
`dictionary`	`QMap<QString, char>`		Словарь кодировок
`dirName`	`QString`		Имя папки
`endCode`	`QString`		Код маркера конца архива
`outpPath`	`QString`		Путь к папке, в которую необходимо записать файл
`longestCodeSize`	`int`		Длина самого длинного кода в словаре. Нужна для определения повреждений в архиве
Public методы класса			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
1	2	3	4
`FileDecoder`	-	-	Конструктор разархиватора, инициализирует `ReadBuffer input`
`dearchive`	`QString path, QString fileName`	-	Распаковывает архив
`decodeDictionary`	-	`bool`	Вернет истину, если файл не пуст
`decodeFile`	`QFile& outf`	-	Распаковывает сжатую запись в файл

Продолжение таблицы 3.2.9

1	2	3	4
readDirectoryTree	-	-	Читает метаданные из архива и создает необходимые папки
getNum()	-	int	Читает число из файла
getPath()	-	QByteArray	Читает строку из файла
toCode	char c, int length	QString	Преобразовывает байт длиной не более восьми в строку нулей и единиц

3.2.10 Класс FileTranslator

Предназначен для сжатия входного файла или файлов в один архив, сохраняя при этом информацию о папках и директориях, если они есть.

Таблица 3.2.10 – Класс FileTranslator

Private поля класса		
Имя	Тип	Описание
fout	QFile	Файл архива
fin	QFile	Текущий входной файл
allFiles	QList<FileEntry>	Список всех файлов на архивацию
coder	Coder	Кодировщик для сжатия информации

Продолжение таблицы 3.2.10

Public методы класса			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
`FileTranslator`	`QString path`	-	Конструктор переводчика файла
`~FileTranslator`	-	-	Закрывает входной и выходной файлы
`openFile`	`QString path_`	-	Открывает указанный в аргументе метода файл
`setAllFiles`	`QList<FileEntry> allFiles_`	-	Устанавливает значение поля `allFiles`
`getAllFiles`	-	`QList<FileEntry>`	Возвращает значение поля `allFiles`
`translateFiles`	-	-	Последовательно архивирует каждый элемент списка `allFiles`
`translateDictionary`	-	-	Записывает словарь текущего файла в архив
`clear`	-	-	Очищает поля объекта и закрывает открытые файлы

3.2.11 Класс MainWindow

Класс основного окна приложения. В нем осуществляются все взаимодействие с пользователем, обработка сигналов и создание потоков архивации и разархивации.

Таблица 3.2.11 – Класс MainWindow

Private поля класса		
Имя	Тип	Описание
1	2	3
ui	Ui::MainWindow*	Поле графической модели программы
model	QFileSystemModel*	Модель файловой системы дерева каталогов
activeFile	QFileInfo*	Текущий выбранный файл
dw	DialogWindow*	Всплывающее окно ввода имени архива
archiveWait	WaitBox*	Всплывающее окно ожидания конца архивации
dearchiveWait	WaitBox*	Всплывающее окно ожидания конца разархивации
at	ArchivationThread*	Поток архивации
dt	DearchivationThread*	Поток разархивации
fileName	QString	Имя архива
MAX_DEPTH	const int	Максимально разрешенное количество файлов в папке для удаления

Продолжение таблицы 3.2.11

1	2	3
processing	bool	Флаг, указывающий на то, производятся в данный момент операции создания или распаковки архива или нет
Public методы класса		
Имя	Принимаемые параметры	Возвращаемое значение	Описание
MainWindow	QWidget* parent = nullptr	-	Конструктор главного окна, инициализи-рует поля класса
~MainWindow	-	-	Деструктор
Private слоты		
Имя	Принимаемые параметры	Возвращаемое значение	Описание
1	2	3	4
on_treeView_clicked	const QModelIndex& index	-	Слот для сигнала о выборе файла в дереве файлов
on_archiveButton_clicked	-	-	Слот для сигнала о нажатии на кнопку архивации

Продолжение таблицы 3.2.11

1	2	3	4
on_dearchiveButton_clicked	-	-	Слот для сигнала о нажатии на кнопку разархивации
on_fileName_entered	QString input	-	Слот для сигнала об окончании ввода имени архива
when_archivation_complete	-	-	Слот сигнала о конце операции архивации
when_archivation_canceled	-	-	Слот сигнала о прерывании операции архивации
when_dearchivation_complete	-	-	Слот сигнала о конце операции разархивации
when_dearchivation_canceled	-	-	Слот сигнала о прерывании операции разархивации
on_delete_button_clicked	-	-	Слот сигнала нажатия на кнопку удаления
when_thread_exception_handled	-	-	Слот для сигнала об исключении в потоке

Продолжение таблицы 3.2.11

Сигналы			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
`close_waitBox`	-	-	Сигнал закрытия окна ожидания конца операции

3.2.12 Класс ReadBuffer

Класс для чтения архива. Его удобство в том, что он считывает буфер информации, размером, указанном в конструкторе класса, и потом дает возможность получать доступ к отдельным байтам буфера.

Таблица 3.2.12 – Класс ReadBuffer

Private поля класса		
Имя	Тип	Описание
`fin`	`QFile`	Входной файл для считывания буфера
`buffer`	`char*`	Массив для чтения
`inputSize`	`const size_t`	Размер массива
`index`	`size_t`	Индекс в массиве
`readen`	`size_t`	Число прочитанных в последней операции чтения из файла байтов

Продолжение таблицы 3.2.12

Public методы класса			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
`ReadBuffer`	`size_t inputSize_`	-	Конструктор буфера чтения, инициализирует массив `buffer`
`~ReadBuffer`	-	-	Деструктор буфера чтения
`openFile`	`QString fileName`	-	Открывает файл
`get`	-	`char`	Возвращает следующий байт файла
`peek`	-	-	Демонстрирует следующий байт файла
`nextBuffer`	-	-	Читает следующий буфер из файла
`isEnd`	-	`bool`	Вернет истину, если указатель файла на конце

3.2.13 Класс TreeFormer

Файл предназначен для формирования бинарного дерева поиска из словаря информации о всех уникальных байтах исходного файла и числа их повторений.

Таблица 3.2.13 – Класс TreeFormer

Private поля класса		
Имя	Тип	Описание
`nodes`	`priority_queue<Node<int, char>*, vector<Node<int, char>*>, NodeComparator>`	Очередь с приоритетом, с помощью которой строится дерево

Продолжение таблицы 3.2.13

Public методы класса			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
`TreeFormer`	-	-	Конструктор формировщика дерева
`TreeFormer`	`QMap<char, int> cat`	-	Конструктор формировщика дерева с добавлением каталога в узлы
`add`	`Node<int, char>* nw`	-	Добавляет узел в очередь
`take`	-	`Node<int, char>* nw`	Берет узел из вершины очереди
`formBTree`	-	`Node<int, char>* nw`	Формирует бинарное дерево и возвращает указатель на вершину

3.2.14 Класс WaitBox

Класс представляет окно ожидания окончания операции, в котором пользователь может нажать на кнопку `Cancel` и прервать текущую операцию сжатия или распаковки.

Таблица 3.2.14 – Класс WaitBox

Private поля класса		
Имя	Тип	Описание
`msgLabel`	`QLabel*`	Строка с информацией для пользователя
`cancel`	`QPushButton*`	Кнопка для отмены текущей операции

Продолжение таблицы 3.2.14

Public методы класса			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
`WaitBox`	`QWidget* parent = nullptr`	-	Конструктор окна ожидания
`show`	-	-	Метод отображения окна на экране
Сигналы			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
`operation_canceled`	-	-	Сигнал, уведомляющий об отмене операции
Public слоты			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
`on_cancel_button_clicked`	-	-	Слот обработки нажатия кнопки отмены
`on_signal_to_close`	-	-	Слот для сигнала о закрытии окна

3.2.15 Класс ArchivationThread

Данный класс необходим для выделения процесса архивации в отдельный поток. В нем происходит сбор информации о папках и передача этой информации классу `FileTranslator`. Также поток перехватывает исключения и передает информацию о них основному потоку с помощью сигнала.

Таблица 3.2.15 – Класс ArchivationThread

Private поля класса		
Имя	Тип	Описание
fileName	QString	Имя архива
activeFile	QFileInfo*	Указатель на выбранный файл
fcList	QList<FileEntry>	Список файлов в папке
ft	FileTranslator	Кодировщик файлов

Public методы класса			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
ArchivationThread	QString filename_, QFileInfo* activeFile_	-	Конструктор потока архивации
setFileName	QString fileName_	-	Устанавливает значение поля fileName
setActiveFile	QFileInfo*	-	Устанавливает значение поля activeFile
clear	-	-	Очищает все поля объекта
getFileName	-	QString	Возвращает значение поля fileName
getFcList	-	QList<FileEntry>	Возвращает значение поля fcList
getArchiveName	-	QString	Возвращает имя архива
run	-	-	Метод для исполнения потока

Продолжение таблицы 3.2.15

Сигналы			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
`exception_executed`	`QString e`	-	Сигнал для главного потока с информацией об исключении

3.2.16 Класс DearchivationThread

Класс необходим для запуска процесса распаковки архива в отдельном потоке. В этом классе происходит перехват исключений из класса `FileDecoder` и перенаправление их в главный поток с помощью сигнала.

Таблица 3.2.16 – Класс DearchivationThread

Private поля класса		
Имя	Тип	Описание
`fileName`	`QString`	Имя архива
`filePath`	`QString`	Путь к архиву

Public методы класса			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
`DearchivationThread`	`QString filename_, QString filePath_`	-	Конструктор потока разархивации
`setFileName`	`QString fileName_`	-	Устанавливает значение поля filrName
`setFilePath`	`QString filePath_`	-	Устанавливает значение поля filePath
`run`	-	-	Метод для исполнения потока

Продолжение таблицы 3.2.16

Сигналы			
Имя	Принимаемые параметры	Возвращаемое значение	Описание
`exception_executed`	`QString e`	-	Сигнал для главного потока с информацией об исключении

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

В данном разделе рассматриваются схемы алгоритмов, используемых в программе.

4.1 Алгоритмы по шагам

4.1.1 Алгоритм записи в архив

Для алгоритма по шагам рассмотрен метод `getNextCodeBuffer` класса `Coder`.

Шаг 1. Начало.

Шаг 2. Входные данные: `QByteArray source`, массив байтов, содержащий информацию из исходного файла, словарь кодировок `QMap<char, QString> dictionary`, где содержится битовая кодировка каждого встречающегося в исходном файле байта, переменная `bool leftPrev`, которая указывает, остался ли незаполненный байт с предыдущего вызова метода, `short prevSize`, которая содержит длину битовой записи в незаконченном байте, `char prev`, где содержится незаконченный байт, переменная `int len`, в которой хранится число битов в массиве `QByteArray temp`, который является внутренней переменной алгоритма.

Шаг 3. Выходные данные: массив `QByteArray buf`, в котором находится сжатая запись полученного массива `source`.

Шаг 4. Начинается итерация по массиву `source`.

Шаг 5. Байт массива кодируется методом `QByteArray encode(char sb)`, где `sb` – байт массива `source`, для которого происходит текущий шаг итерации.

Шаг 6. Переменная `len` увеличивается на длину кодировки байта в словаре `dictionary`.

Шаг 7. Итерация по переменной `len`.

Шаг 8. Если переменная `len` больше либо равна восьми, то первый байт массива `temp` копируется в выходной массив `buf`.

Шаг 9. Декремент переменной `len` на 8

Шаг 10. Конец итерации по переменной `len`.

Шаг 11. Проверяется переменная `len`, если она не ноль, то оставшийся в массиве `temp` незаконченный байт сохраняется в переменную `prev`, число битов в этом байте записывается в `prevSize`, а флаг `leftPrev` переходит в значение истина.

Шаг 12. Конец итерации по массиву `source`.

Шаг 13. Вернуть `buf`.

Шаг 14. Конец.

4.1.2 Алгоритм формирования дерева байтов

Для алгоритма по шагам рассмотрен метод `formBTree` класса `TreeFormer`.

Шаг 1. Начало.

Шаг 2. Входные данные: очередь с приоритетом `priority_queue<Node<int, char>*, vector<Node<int, char>*>, NodeComparator> nodes`, в которой узлы дерева сортируются при помощи объекта-компаратора так, что в начале очереди будет находится узел с наименьшим значением поля `count`.

Шаг 3. Выходные данные: указатель на вершину дерева `Node<int, char>* tree`.

Шаг 4. Если очередь пуста, вернуть `nullptr`.

Шаг 5. Создание нового узла с нулевым значением поля `count`, который будет использован для получения уникального бинарного кода конца архива. Таким образом узел со значением count равно нулю попадет в начало очереди, и для него сгенерируется максимально длинный бинарный код.

Шаг 6. Итерация по очереди `nodes` пока размер очереди не станет равен единице.

Шаг 7. Создание нового узла из двух первых в очереди и удаление этих двух узлов из очереди, значение `count` нового узла является суммой полей `count` его наследников.

Шаг 8. Помещение только что созданного узла в очередь.

Шаг 9. Конец итерации по очереди `nodes`.

Шаг 10. Значение узла `tree` равняется последнему узлу очереди

Шаг 11. Вернуть `tree`.

Шаг 12. Конец.

5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

В данном разделе описывается функциональное тестирование программы.

5.1 Пользовательский ввод

В программе производится проверка пользовательского ввода в разных ее вариациях. Для начала работы необходимо выбрать файл или директорию, кликнув по нему (ней) левой кнопкой мыши. Если файл не был выбрал, то кнопки архивации, разархивации и удаления будут недоступны (рисунок 5.1.1).

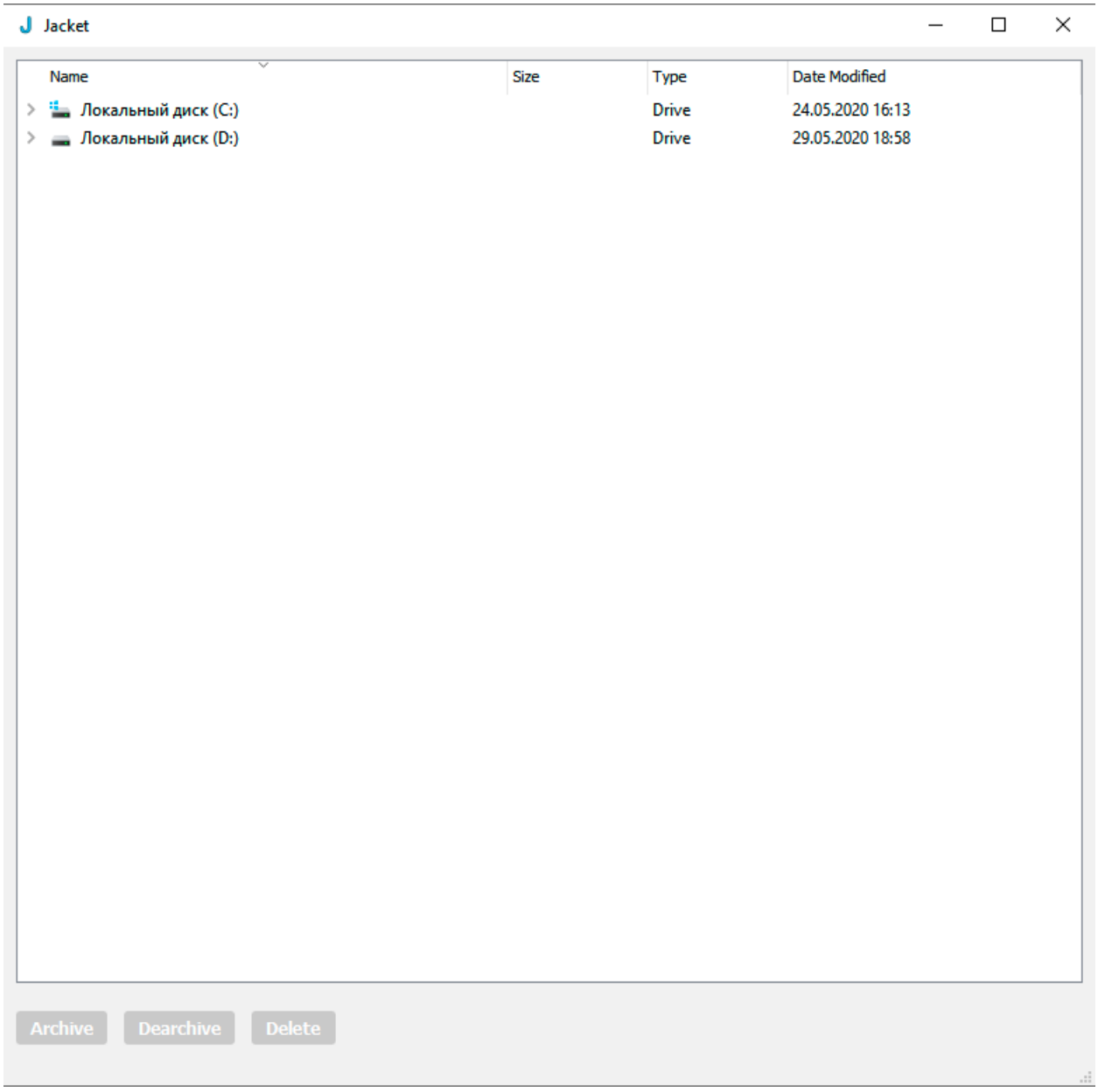

Рисунок 5.1.1

Если пользователь пытается заархивировать Диск, выводится предупреждение (рисунок 5.1.2).

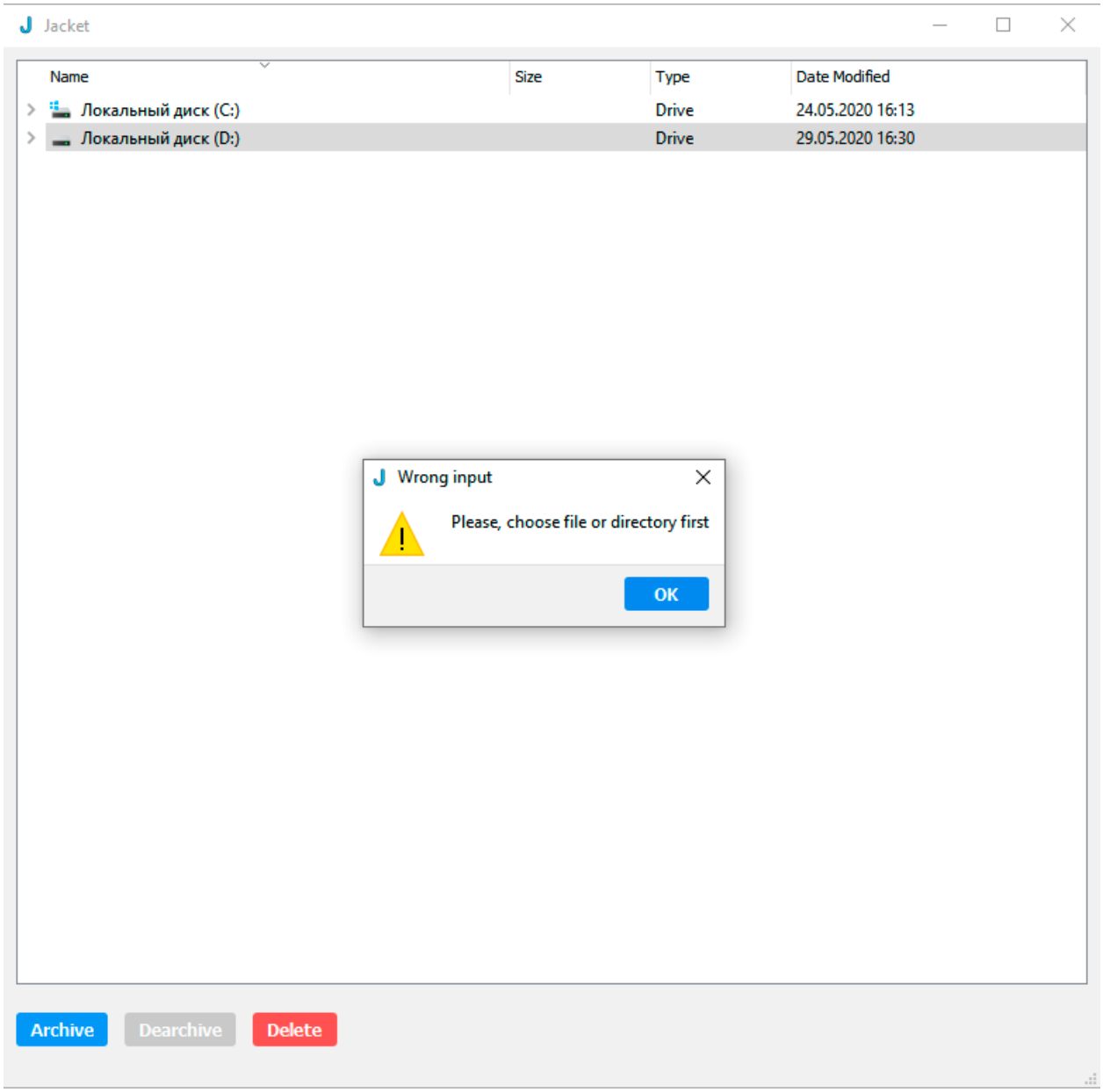

Рисунок 5.1.2

При нажатии на кнопку Delete пользователь получает всплывающее окно, уточняющее его выбор (рисунок 5.1.3).

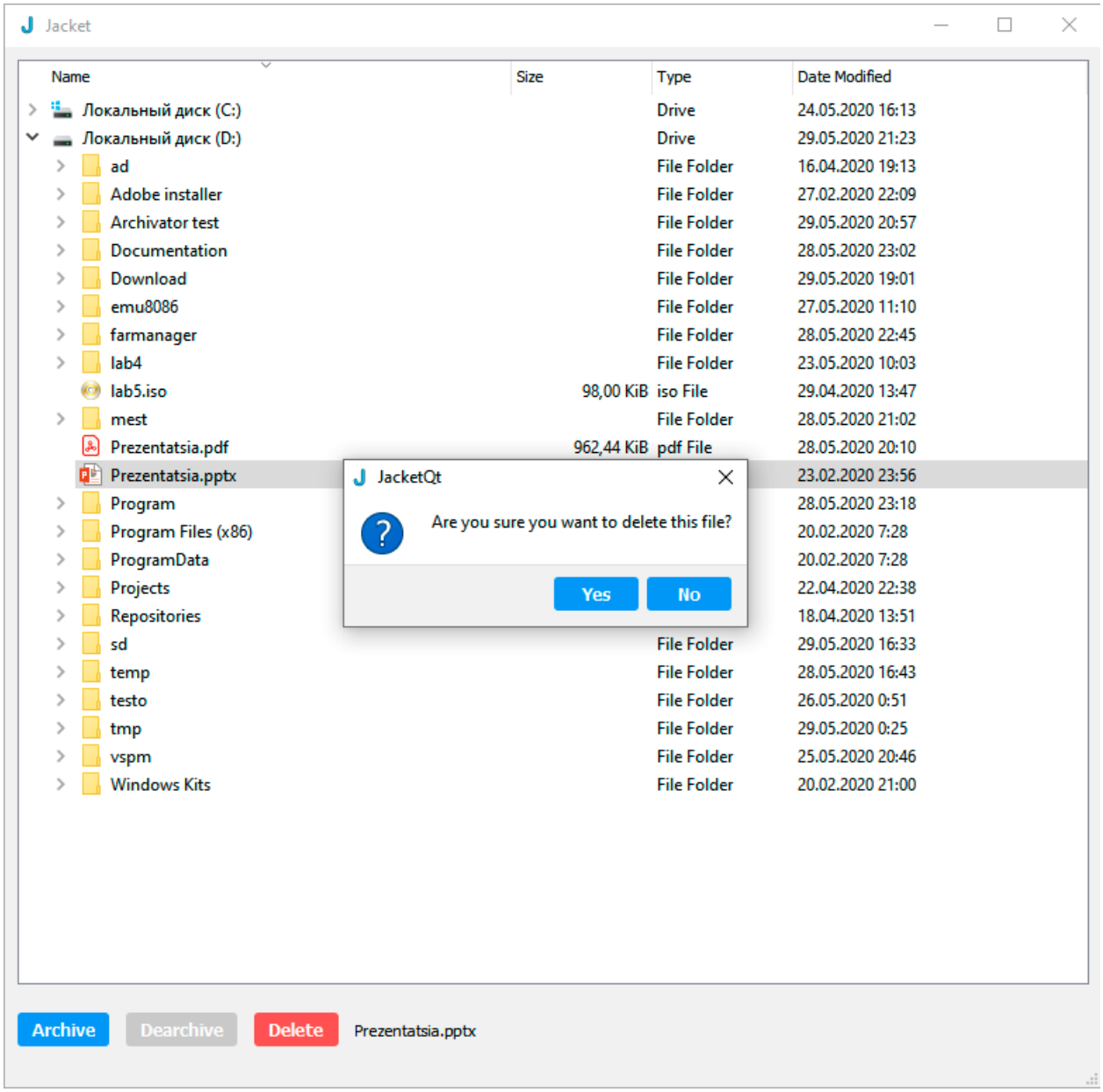

Рисунок 5.1.3

При выборе файла и нажатии кнопки Archive выводится окно ввода имени файла (рисунок 5.1.4). В нем осуществляется проверка ввода. Программа выводит сообщение об ошибке при вводе неправильного имени файла (рисунок 5.1.5).

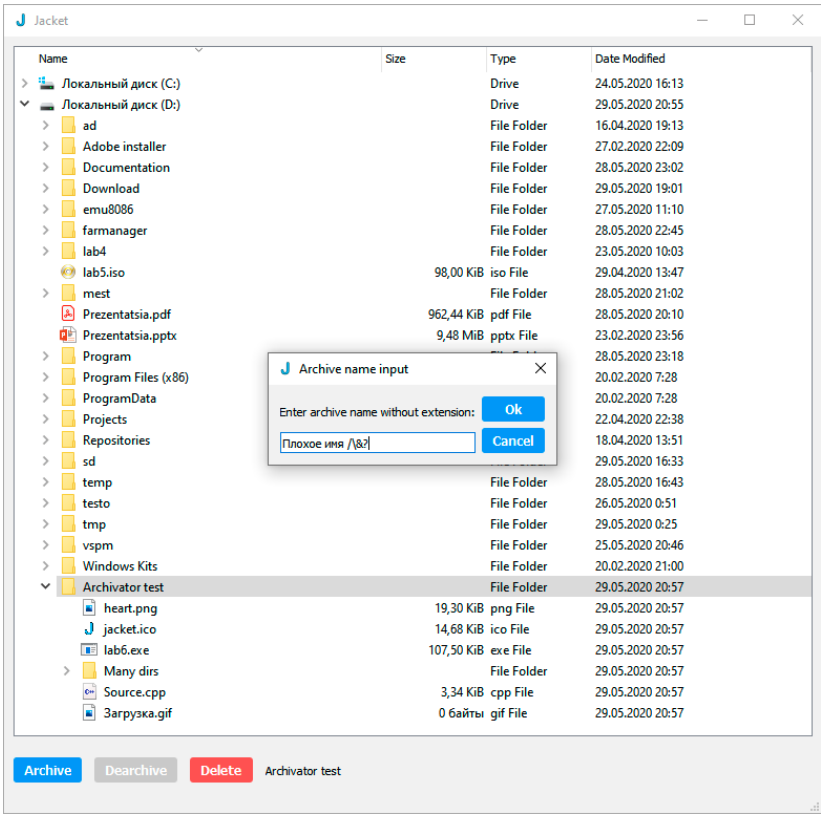

Рисунок 5.1.4

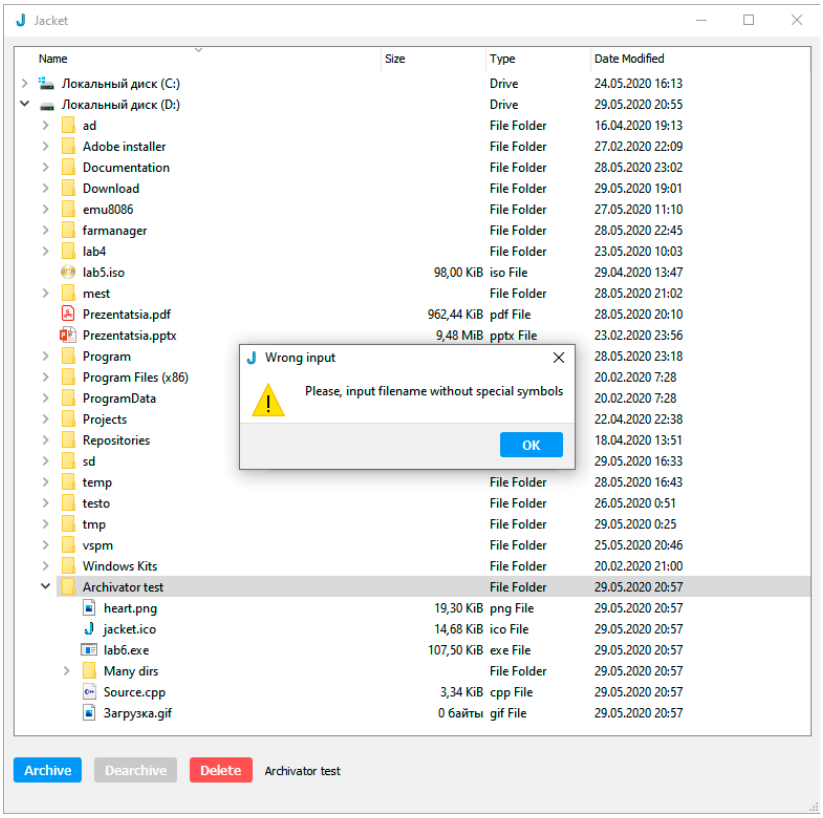

Рисунок 5.1.5

5.2 Внутренние ошибки

Также обрабатываются внутренние ошибки работы программы. Если в ходе работы поток архивации или разархивации сталкивается с проблемой, он уведомляет главный поток об ошибке, используя систему слотов и сигналов. Например, с такой ошибкой можно столкнуться, если специально повредить архив (рисунок 5.2.1, рисунок 5.2.2), а затем попытаться его разархивировать (рисунок 5.2.3).

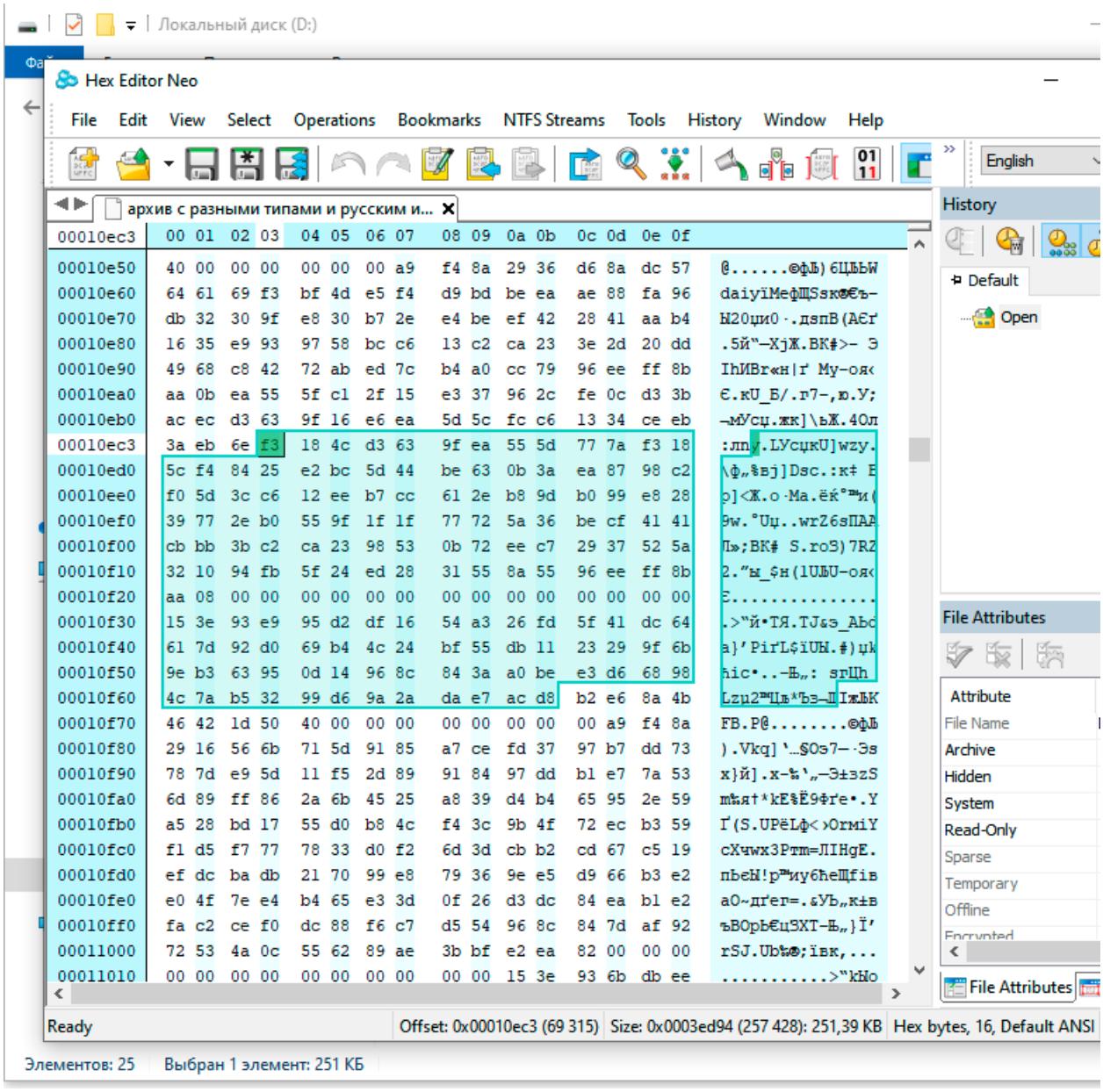

Рисунок 5.2.1

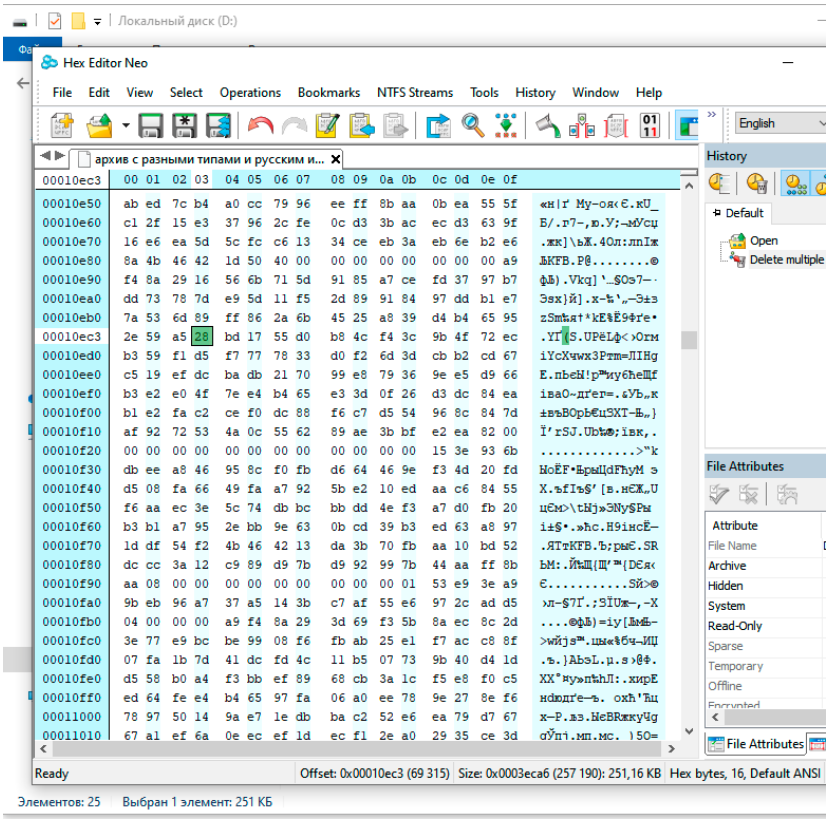

Рисунок 5.2.2

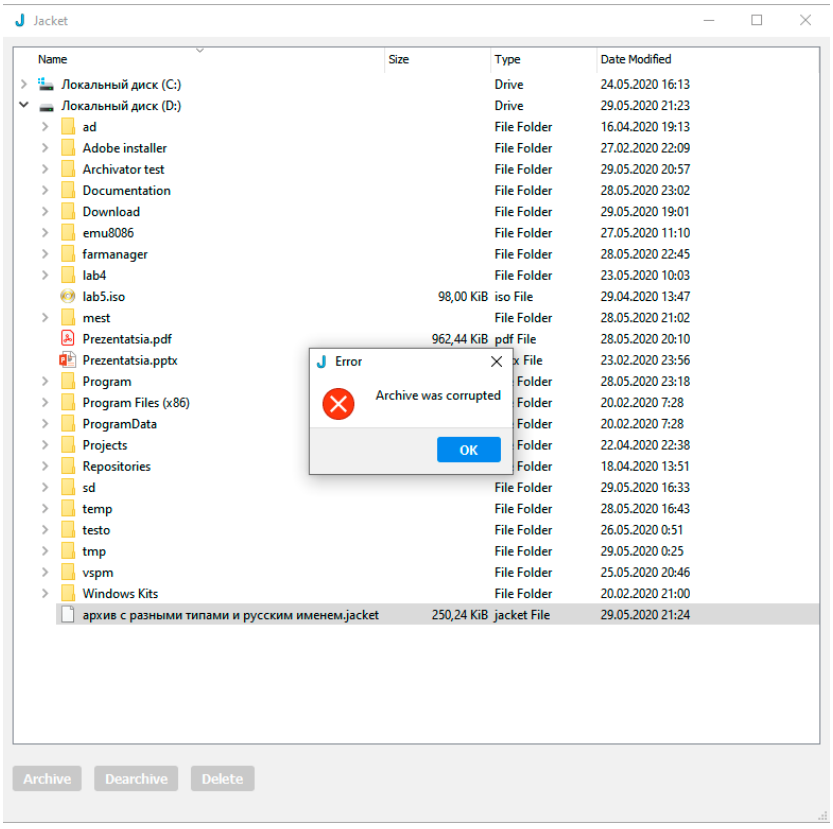

Рисунок 5.2.3

Таким образом обрабатываются и другие возможные ошибки, такие как ошибка открытия файла, ошибка создания файла, ошибка создания директории и некоторые логические ошибки (например, попытка использования неинициализированной переменной).

5.3 Тестирование архивации данных

Для тестирования архивации выбирается директория, содержащая множество вложенных папок и файлов, разных типов (рисунок 5.3.1). На рисунке 5.3.2 виден только что созданный файл архива, еще пустой. Также на том же рисунке видно всплывающее окно ожидания завершения текущей операции, при нажатии на кнопку Cancel текущая операция прервется, а незавершенный архив удалится. На рисунке 5.3.3 демонстрируется сравнение размеров исходной директории и архива. Видно, что архив весит на 50 Кбайт меньше. Разница в размере исходного и архивированного файлов зависит от структуры данных в файле. Так наиболее заметна разница в размере при архивации текстовых файлов, и в меньшей мере при архивации изображений и файлов с кодом.

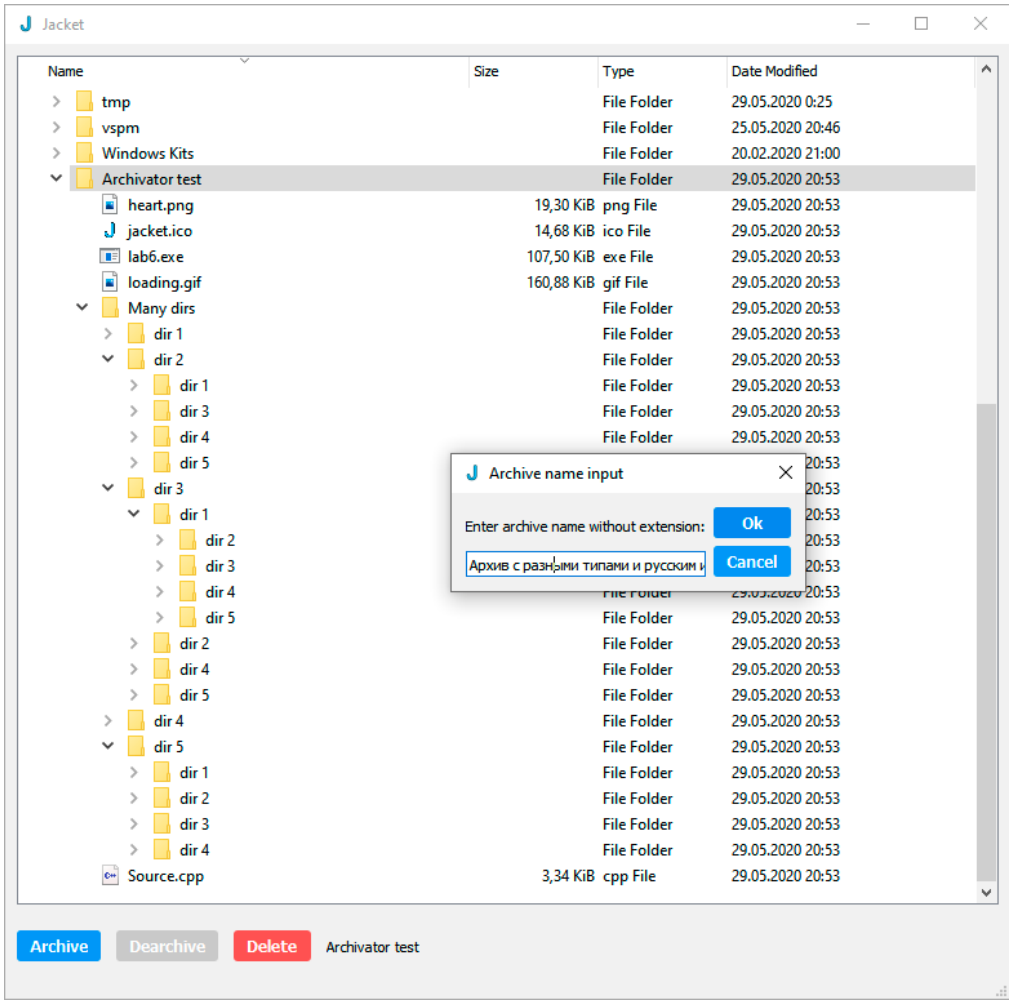

Рисунок 5.3.1

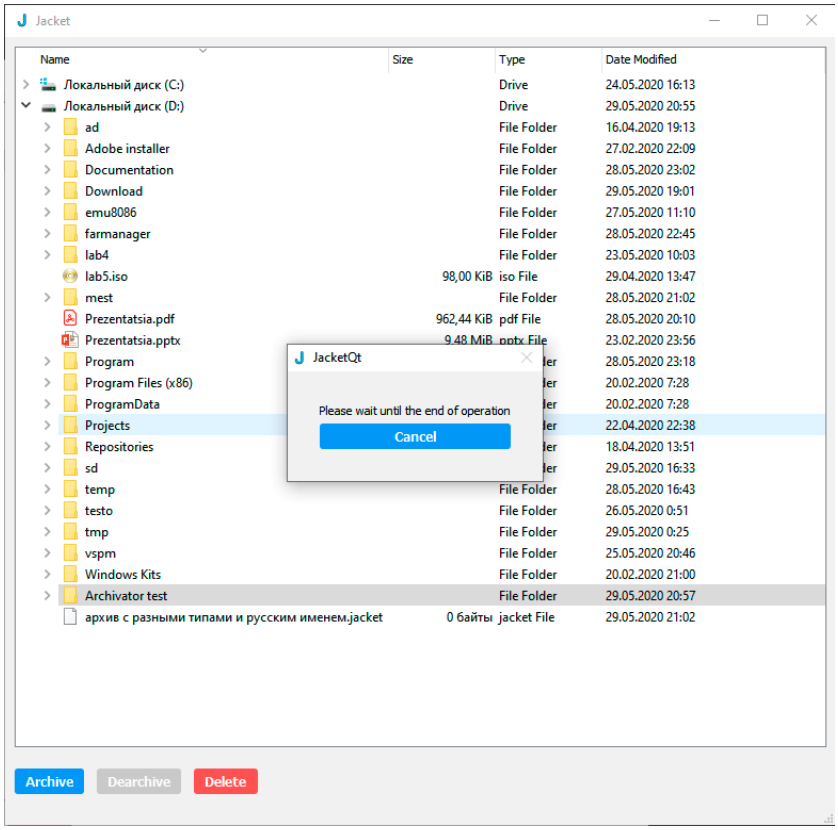

Рисунок 5.3.2

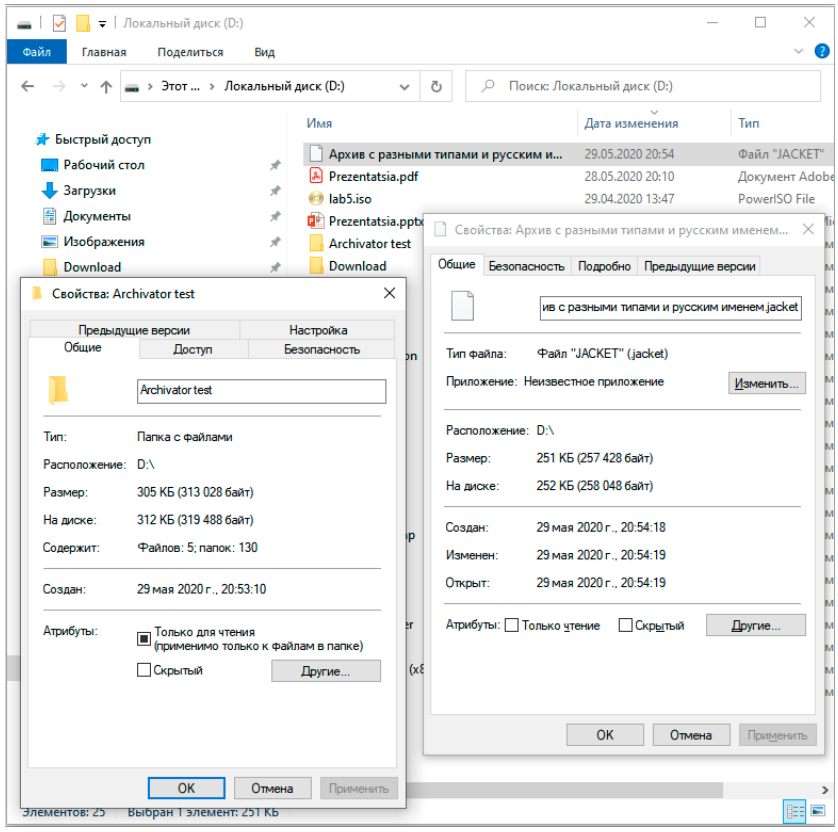

Рисунок 5.3.3

5.4 Тестирование разархивации

Для тестирования разархивации берется архив из предыдущего теста и перемещается в какую-то временную директорию, чтобы при разархивации данные не перезаписали оригинал (рисунок 5.4.1). Далее нажимается кнопка Dearchive и ожидается окончание операции (рисунок 5.4.2). После этого полученные файлы просматриваются на наличие ошибок (рисунок 5.4.3), а также сравниваются размеры директории до архивации и разархивации и после (рисунок 5.4.4).

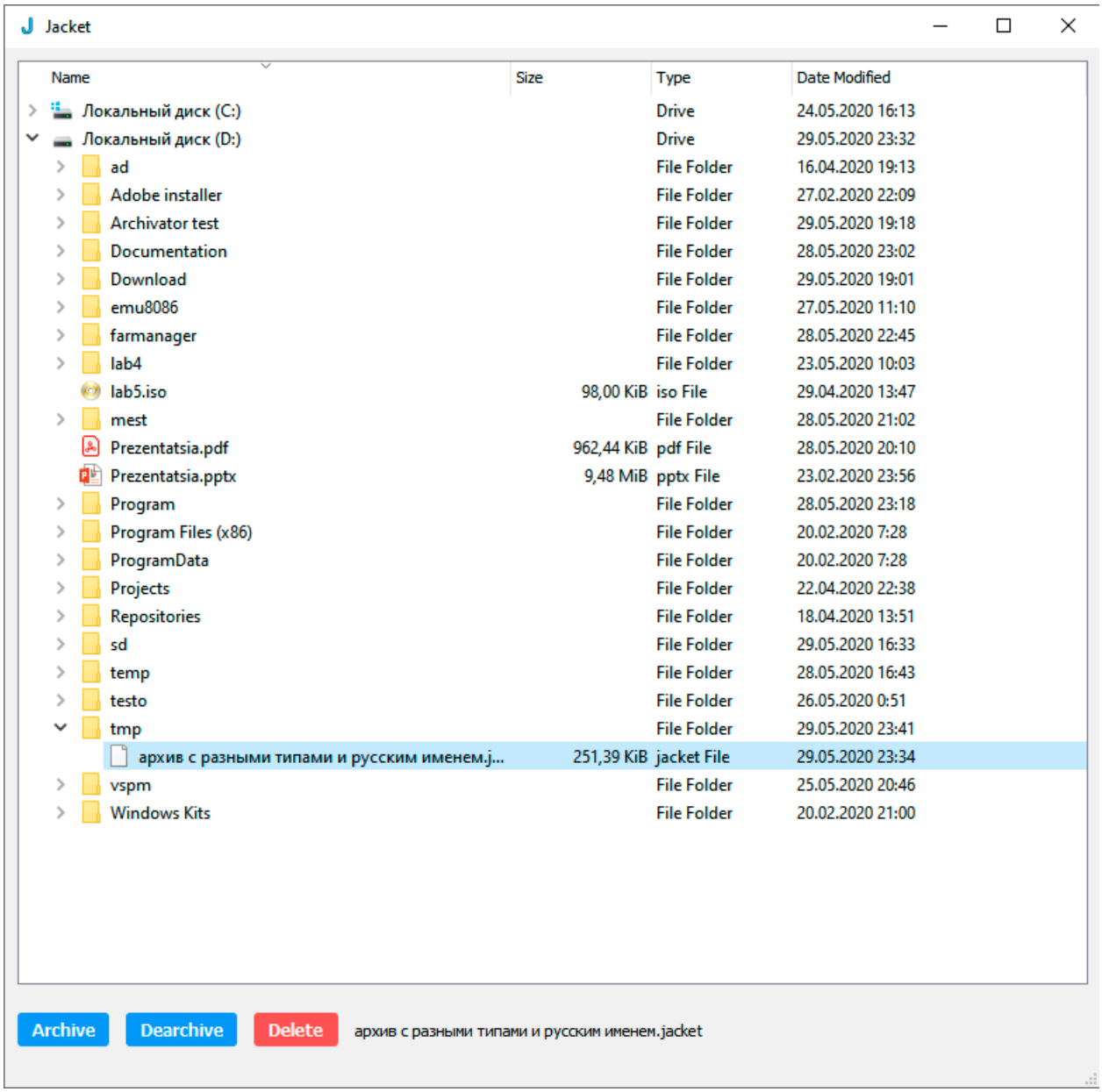

Рисунок 5.4.1

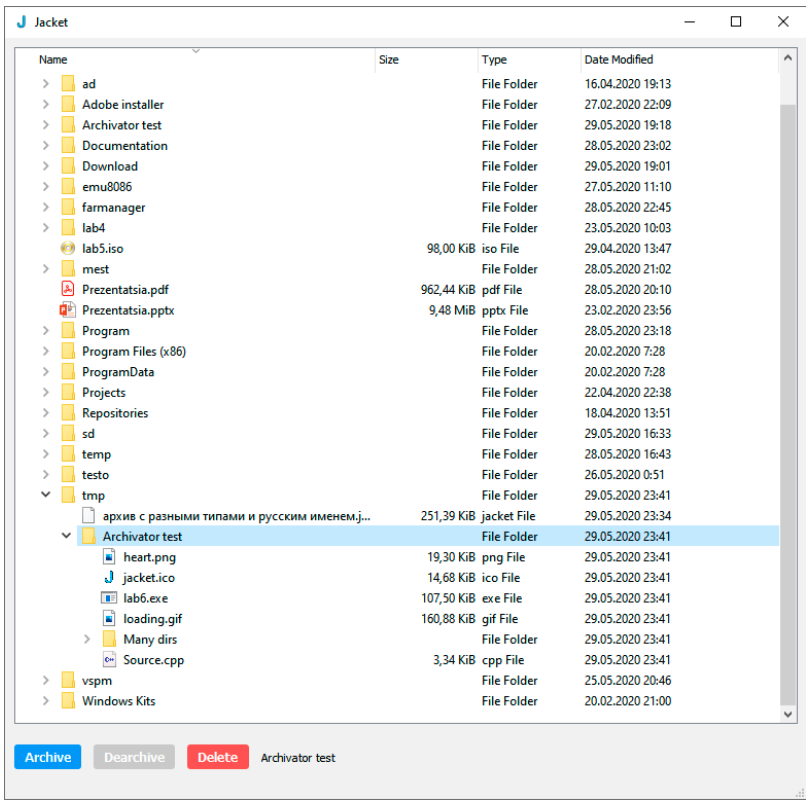

Рисунок 5.4.2

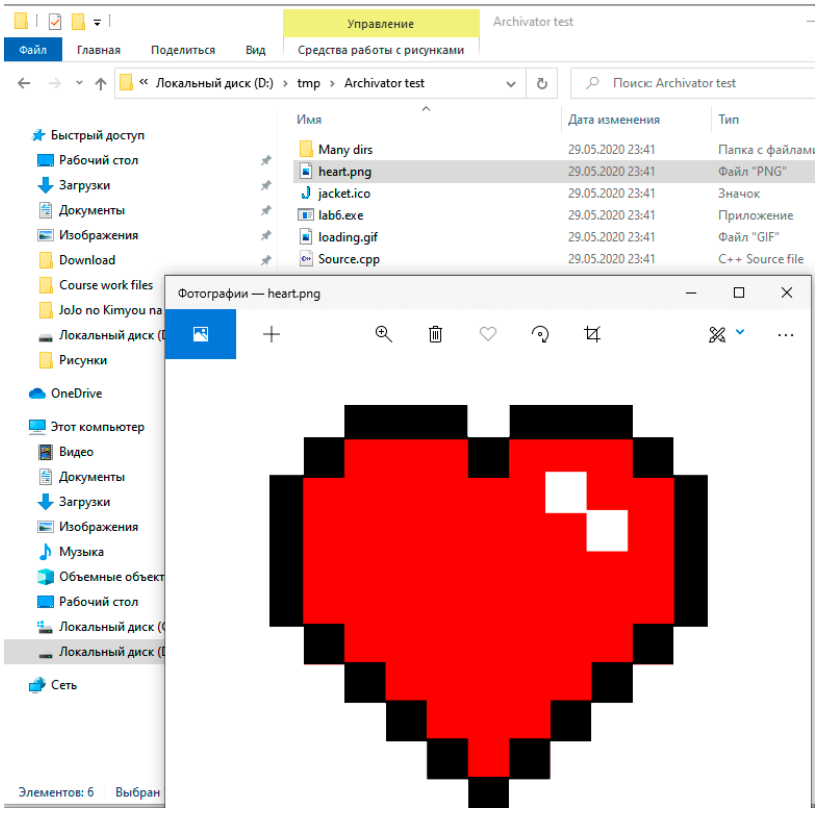

Рисунок 5.4.3

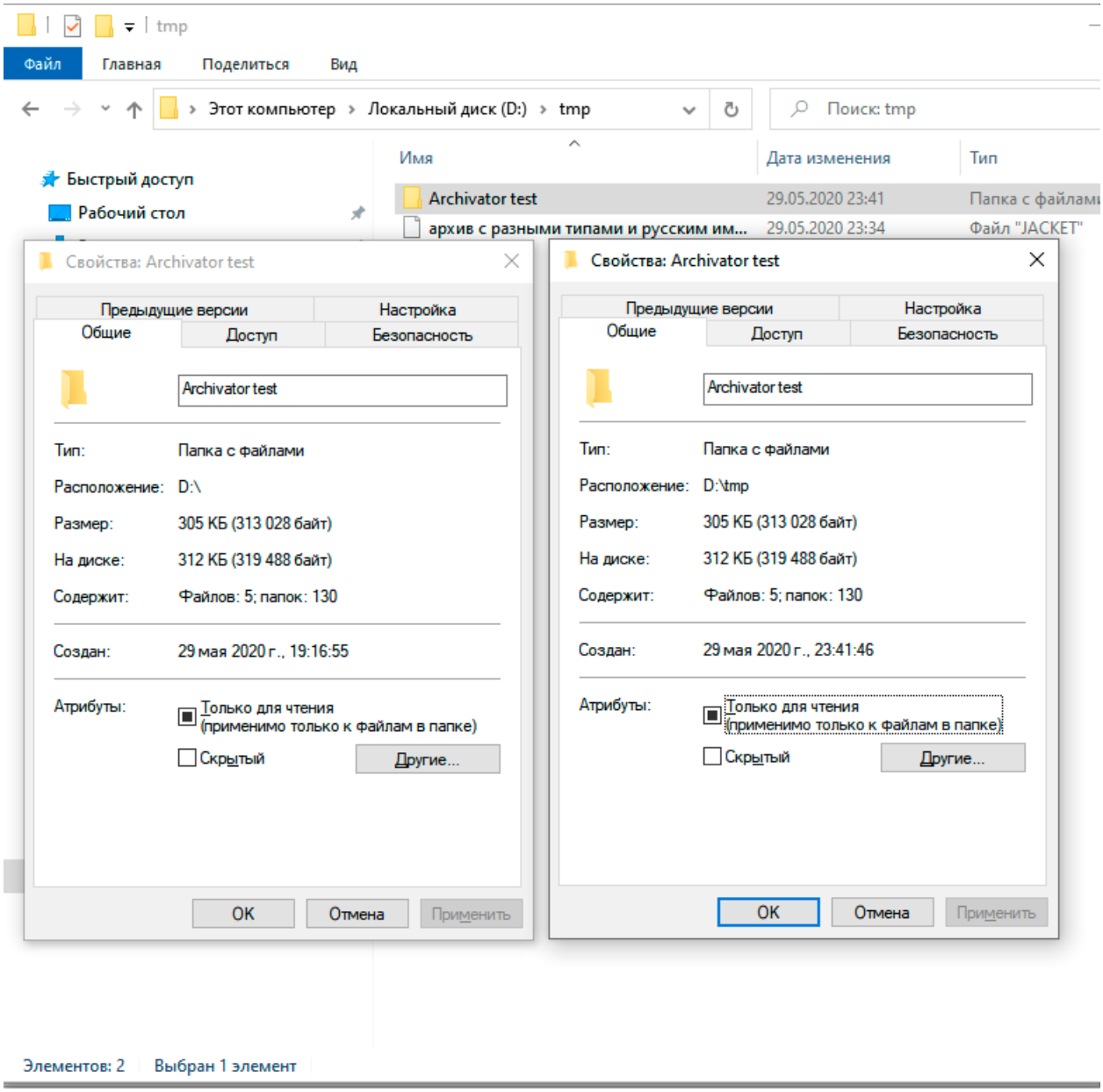
tmp
Файл
Главная
Поделиться
Вид
Этот компьютер › Локальный диск (D:) › tmp
Поиск: tmp
Быстрый доступ
Рабочий стол
Имя
Дата изменения
Тип
Archivator test
29.05.2020 23:41
архив с разными типами и русским им...
29.05.2020 23:34
Файл "JACKET"
Свойства: Archivator test
Предыдущие версии
Настройка
Общие
Доступ
Безопасность
Archivator test
Тип: Папка с файлами
Расположение: D:\
Размер: 305 КБ (313 028 байт)
На диске: 312 КБ (319 488 байт)
Содержит: Файлов: 5; папок: 130
Создан: 29 мая 2020 г., 19:16:55
Атрибуты:
Только для чтения (применимо только к файлам в папке)
Скрытый
Другие...
OK
Отмена
Применить
Свойства: Archivator test
Предыдущие версии
Настройка
Общие
Доступ
Безопасность
Archivator test
Тип: Папка с файлами
Расположение: D:\tmp
Размер: 305 КБ (313 028 байт)
На диске: 312 КБ (319 488 байт)
Содержит: Файлов: 5; папок: 130
Создан: 29 мая 2020 г., 23:41:46
Атрибуты:
Только для чтения (применимо только к файлам в папке)
Скрытый
Другие...
OK
Отмена
Применить
Элементов: 2
Выбран 1 элемент

Рисунок 5.4.4

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

6.1 Системные требования

Данная программа была разработана на операционной системе Windows 10 в среде разработки QT Creator Community edition. Процессор Intel Core i3-7300U, ОЗУ 8ГБ DDR3. Для нормальной работы приложения достаточно 20 МБ свободной оперативной памяти, а также свободное место на диске, достаточное для записи выходного архива или разархивированного содержимого сжатого файла.

6.2 Использование приложения

При запуске приложения выводится главное окно приложения, в котором изначально все кнопки взаимодействия с приложением заблокированы (рисунок 6.2.1). В нем необходимо выбрать файл или директорию. Если выбранный файл имеет расширение “.jacket” кнопка Dearchive станет доступной.

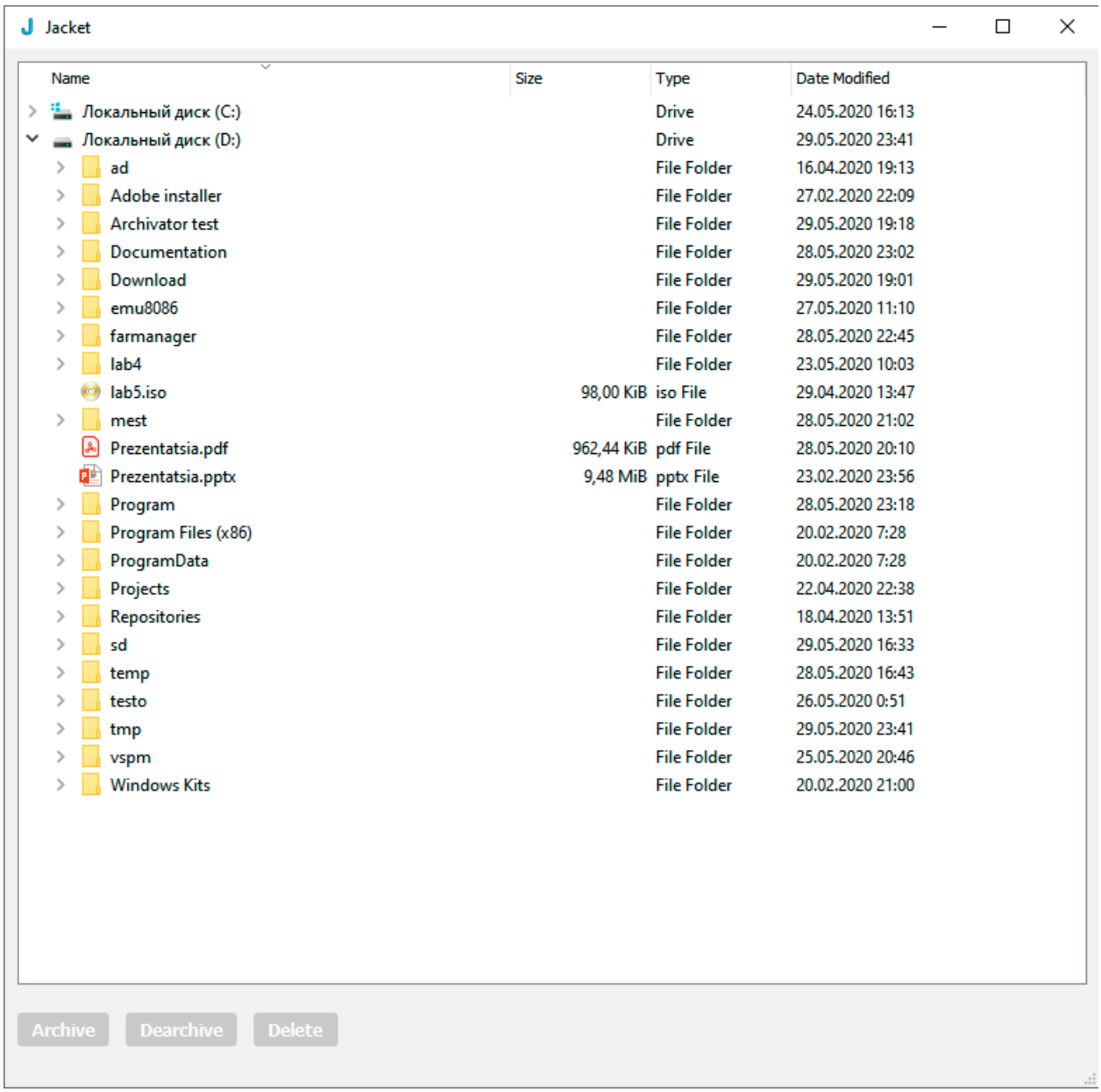

Рисунок 6.2.1

Для взаимодействия с приложением пользователь может использовать кнопки Archive – заархивировать, Dearchive – разархивировать, Delete – удалить. Навигации по файловой системе осуществляется путем кликов по файлам и директориям. Для раскрытия содержимого папки необходимо нажать стрелочку рядом с иконкой папки или дважды кликнуть по папке (рисунок 6.2.2).

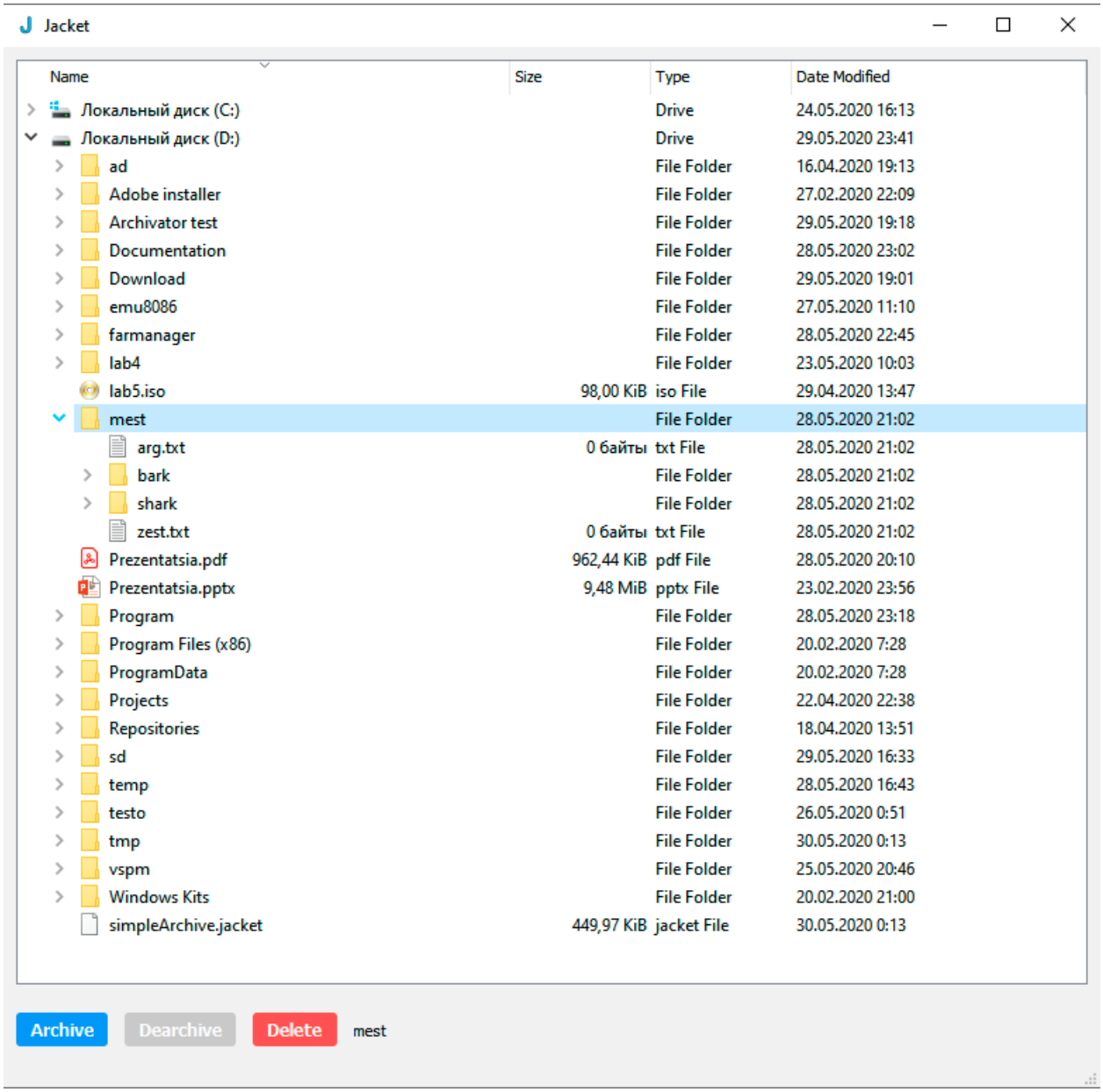

Рисунок 6.2.2

После выбора файла или папки, пользователь может заархивировать его или ее соответственно. Для этого необходимо сначала выделить папку, а затем нажать кнопку Archive, после чего пользователь увидит окно ввода имени архива (рисунок 6.2.3). Пользователь может как ввести имя, так и отказаться от этого. Для отказа нужно кликнуть по кнопке Cancel. Имя файла должно состоять хотя бы из одного символа и подходить под стандартные требования именования файлов в Windows. После ввода имени пользователь должен нажать кнопку Ok. Начинается операция архивации. Пользователь видит окно ожидания конца операции, в котором он может отменить архивацию (рисунок 6.2.4).

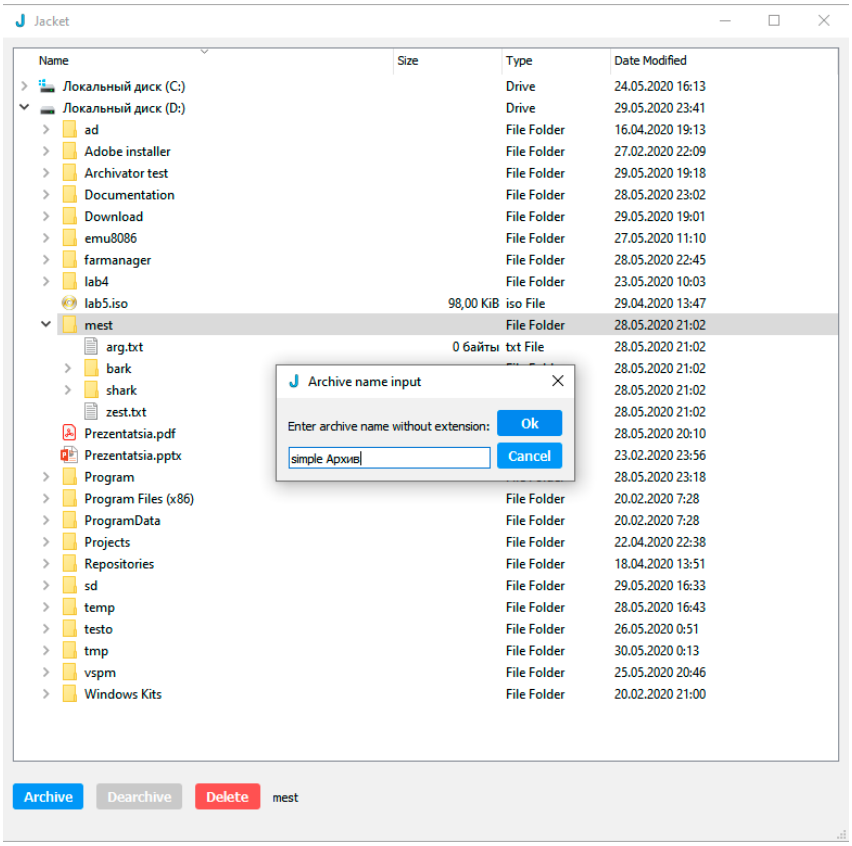

Рисунок 6.2.3

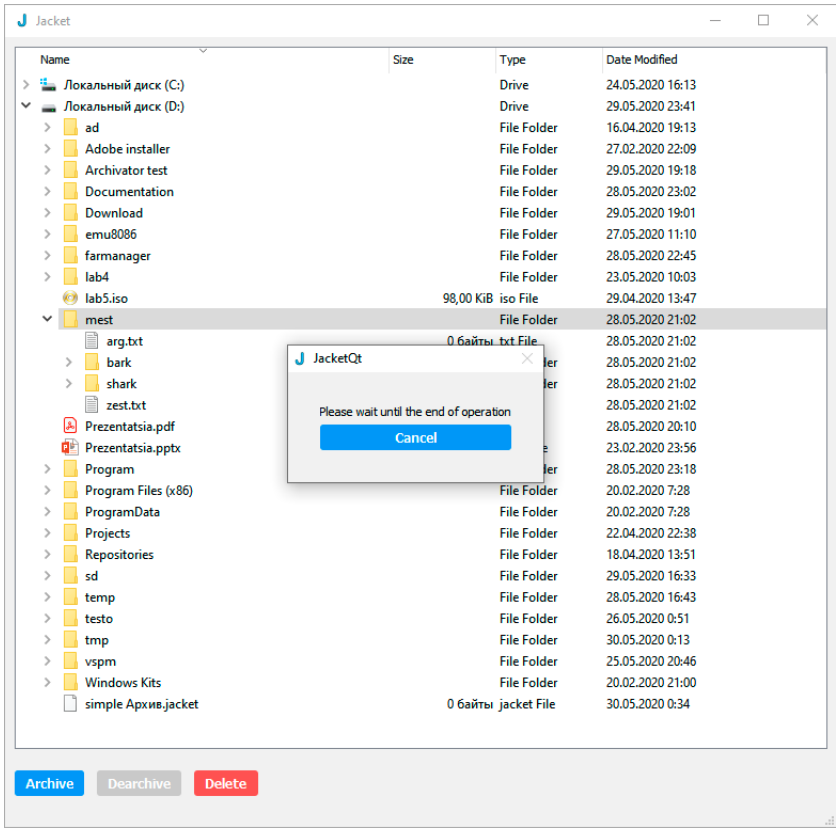

Рисунок 6.2.4

Для разархивации пользователь должен выбрать файл с расширением “.jacket”. Только тогда кнопка Dearchive становится доступна (рисунок 6.2.5). При нажатии на нее начинется процесс разархивации. Как и в случае с архивацией, пользователь видит перед собой окно ожидания конца операции, в котором он может прервать процесс разархивации.

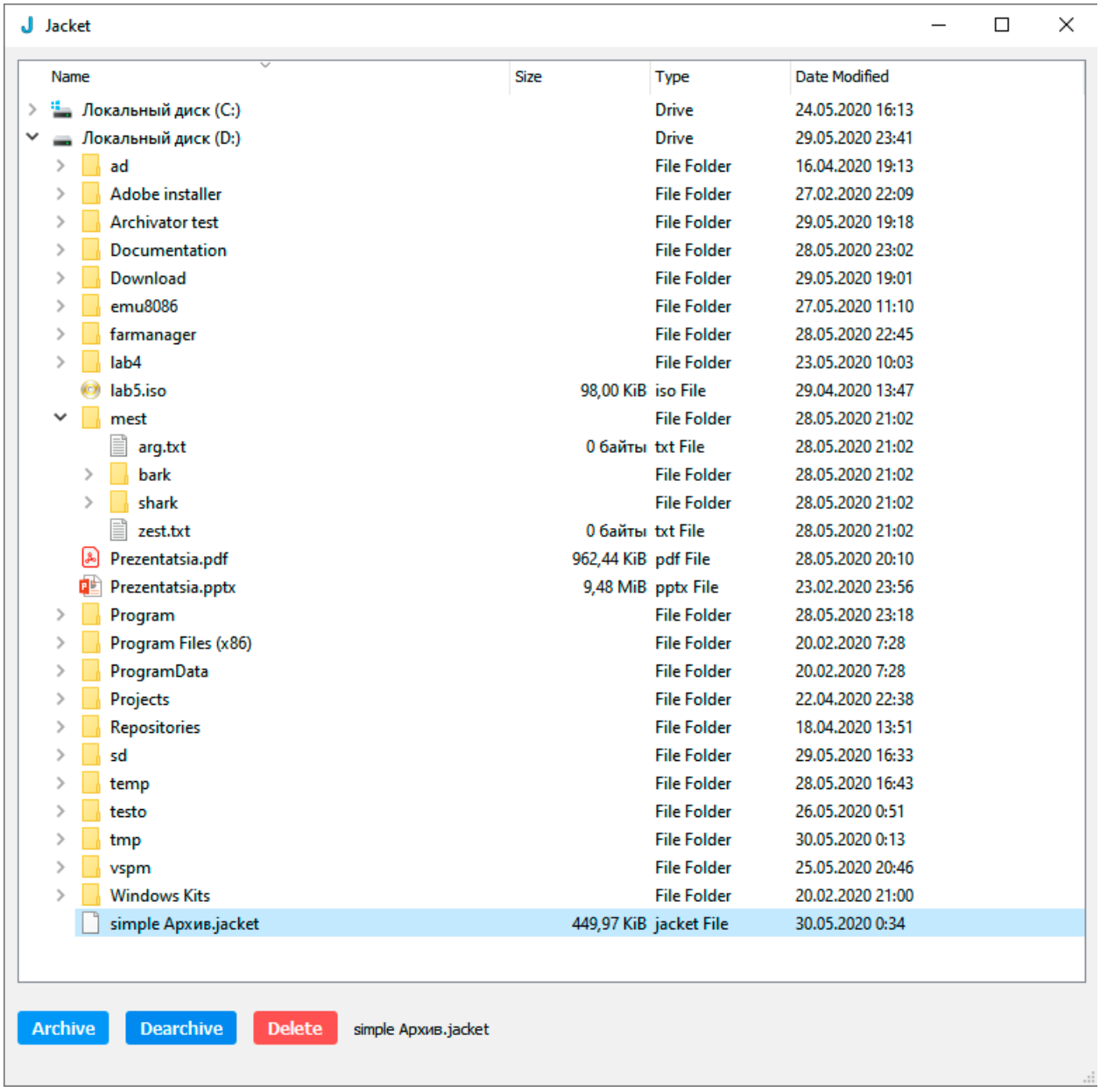

Рисунок 6.2.5

Для удаления файла или директории пользователю необходимо выбрать их, а затем нажать на кнопку Delete. После этого выводится диалоговое окно уточнения, уверен ли пользователь в своем решении удалить файл или папку (рисунок 6.2.6). При нажатии кнопки Yes произойдет удаление выбранного файла или директории.

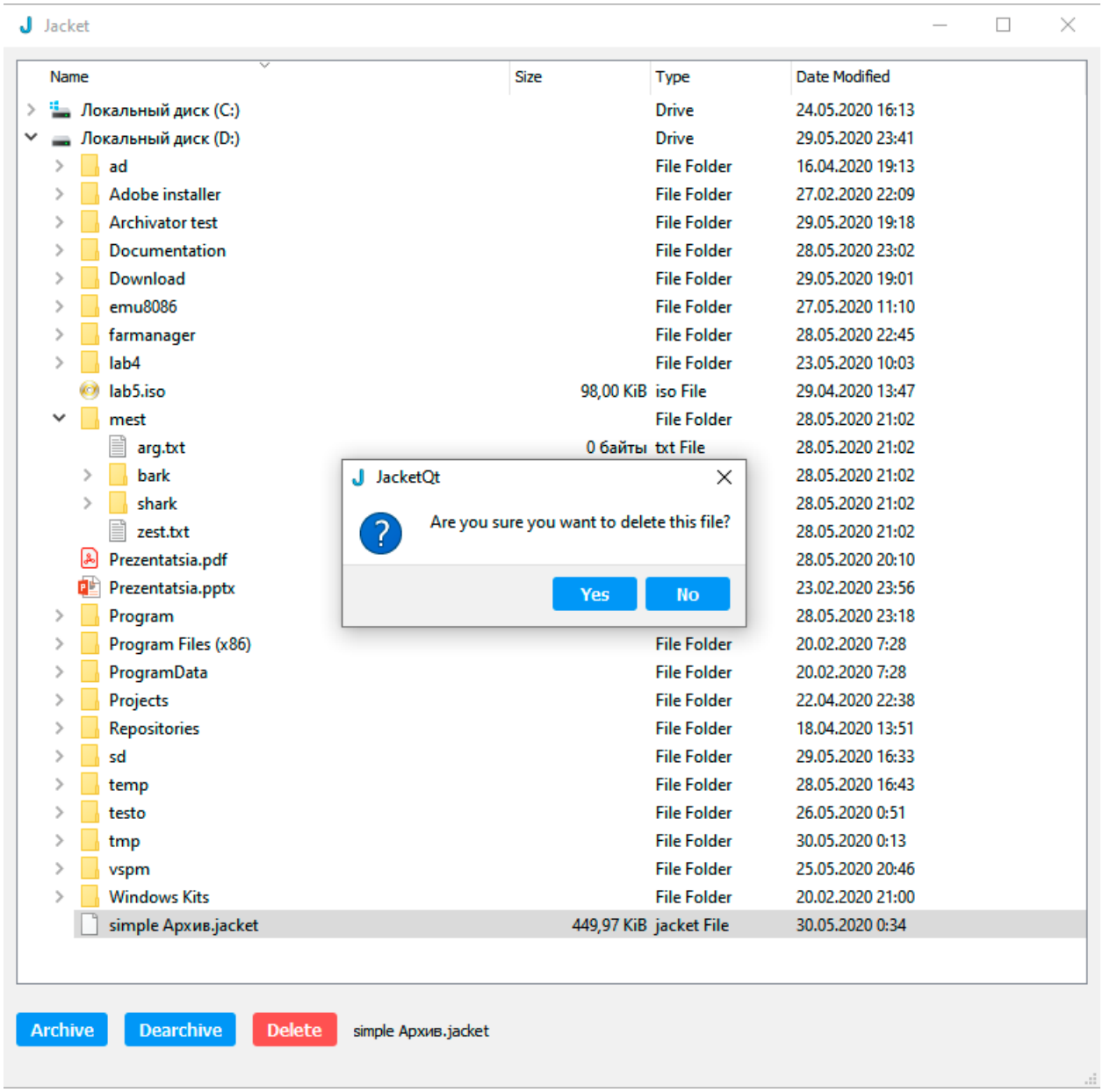
Jacket
Name
Size
Type
Date Modified
Локальный диск (C:)
Drive
24.05.2020 16:13
Локальный диск (D:)
Drive
29.05.2020 23:41
ad
File Folder
16.04.2020 19:13
Adobe installer
File Folder
27.02.2020 22:09
Archivator test
File Folder
29.05.2020 19:18
Documentation
File Folder
28.05.2020 23:02
Download
File Folder
29.05.2020 19:01
emu8086
File Folder
27.05.2020 11:10
farmanager
File Folder
28.05.2020 22:45
lab4
File Folder
23.05.2020 10:03
lab5.iso
98,00 KiB
iso File
29.04.2020 13:47
mest
File Folder
28.05.2020 21:02
arg.txt
0 байты
txt File
28.05.2020 21:02
bark
28.05.2020 21:02
shark
28.05.2020 21:02
zest.txt
28.05.2020 21:02
Prezentatsia.pdf
28.05.2020 20:10
Prezentatsia.pptx
23.02.2020 23:56
Program
28.05.2020 23:18
Program Files (x86)
File Folder
20.02.2020 7:28
ProgramData
File Folder
20.02.2020 7:28
Projects
File Folder
22.04.2020 22:38
Repositories
File Folder
18.04.2020 13:51
sd
File Folder
29.05.2020 16:33
temp
File Folder
28.05.2020 16:43
testo
File Folder
26.05.2020 0:51
tmp
File Folder
30.05.2020 0:13
vspm
File Folder
25.05.2020 20:46
Windows Kits
File Folder
20.02.2020 21:00
simple Архив.jacket
449,97 KiB
jacket File
30.05.2020 0:34
JacketQt
Are you sure you want to delete this file?
Yes
No
Archive
Dearchive
Delete
simple Архив.jacket

Рисунок 6.2.6

ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта было разработано приложение для архивации и разархивации файлов. Предусмотрены различные возможные проблемы от нехватки места для создания архива или распаковки сжатых данных, до проверки архива на повреждения. Для архивации данных использовался алгоритм Хаффмана, что позволило сжимать данные без потерь. В программе работают три потока: поток архивации, поток разархивации и поток основного окна и пользовательского интерфейса. Для разработки программы был дополнительно изучен фреймворк QT, что позволило создать удобный пользовательский интерфейс, а также избежать создания дополнительного объема кода путем модернизации существующих стандартных классов, ведь существовала возможно использовать стандартные классы QT, соответствующие нуждам.

К достоинствам разработанной программы относятся скорость, легковесность, а также удобный пользовательский интерфейс. Программа работает в нескольких потоках, однако, в целях модернизации и ускорения быстродействия, возможно разделения потоков еще на несколько. Также в сам интерфейс программы можно добавить дополнительную функциональность, такую как: перемещение файлов и директорий, копирование файлов и директорий и другое, а также возможность просмотра содержимого архива без предварительной разархивации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Habr - электронный веб-ресурс в формате коллективного блога посвященный сфере IT и программированию. – Режим доступа: http://habr.com/

[2] StackOverflow - форум, посвященный решению различных вопросов по программированию. – Режим доступа: http:// stackoverflow.com/

[3] Ravesli - образовательный портал, на который выкладываются переводы полезных статей по C++. – Режим доступа : http:// ravesli.com/

[4] Лафоре Р.Объектно-ориентированное программирование в C++:. Питер, 4-е издание, 2015г.

[5] Луцик Ю. А. «Объектно-ориентированное программирование на языке C++: учеб. Пособие» Ю. А. Луцик, В. Н. Комличенко. – Минск: БГУИР, 2008г.

[6] Qt doc - Документация фреймворка Qt. Режим доступа: http://doc.qt.io/

ПРИЛОЖЕНИЕ А
(*обязательное*)

Структурная схема

ПРИЛОЖЕНИЕ Б
(*обязательное*)

Диаграмма классов