Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность 1-40 05 01 03 «Информационные системы и технологии»

Специализация «Издательско-полиграфический комплекс»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА**

по дисциплине «Объектно-ориентированное программирование»

Тема «Файл-менеджер»

Исполнитель

Cтудентка 2 курса 2 группы \_\_\_\_\_\_\_\_\_ П. О. Панченко

подпись, дата

Руководитель

доцент, канд. техн. наук \_\_\_\_\_\_\_\_\_ Н. В. Пацей

подпись, дата

Курсовой проект защищен с оценкой \_\_\_\_\_\_\_\_\_

Руководитель \_\_\_\_\_\_\_\_\_ Н. В. Пацей

подпись

МИНСК 2015

**Оглавление**

+

[Введение 3](#_Toc419833888)

[1 1.Теоретическое введение 4](#_Toc419833889)

[1.1 Объектно-ориентированное программирование 4](#_Toc419833890)

[1.2. Среда Visual Studio.NET 4](#_Toc419833891)

[1.3.  C#  WinForm 5](#_Toc419833892)

[2. Анализ поставленной задачи и формирование требований 6](#_Toc419833894)

[2.1 Отношение к виду 6](#_Toc419833895)

[2.2 Аналоги 6](#_Toc419833896)

[2.3 Сравнение 10](#_Toc419833897)

[2.4 Требования 10](#_Toc419833898)

[3.Диаграмма классов 11](#_Toc419833899)

[4.Руководство пользователя 15](#_Toc419833900)

[5. Описание программы 21](#_Toc419833901)

[5.1 Использование пространств имен Net Framework 21](#_Toc419833902)

[5.2 Реализуемые функции 22](#_Toc419833903)

[6.Тестирование 27](#_Toc419833904)

[Заключение 32](#_Toc419833905)

[Список используемой литературы и источников 33](#_Toc419833906)

# Введение

Со временем на компьютере практически у каждого пользователя накапливается огромное количество файлов. Аккуратные пользователи время от времени проводят уборку своего электронного хлама. Но довольно сложно бывает сразу определить, что собой представляет тот или иной файл, особенно если на компьютере накопилось много различных файлов с невнятными именами.

В этом случае обычно на помощь приходят утилиты Windows, программа Total Commander. Так же в сети Интернет можно найти великое множество любительских программ, написанных самостоятельно и свободных в доступе. Любительские программы пишутся вследствие неудовлетворённости утилитами Windows: неудобством их использования, неполным набором исполняемых задач, а так же для освоения новых возможностей сред программирования Помимо любительских программ, имеются профессиональные, сложные приложения, диапазон возможностей которых гораздо шире. В основном эти программы предназначены для работы с аудио и видеофайлами, т.к. они могут содержать гораздо больше информации о себе, нежели простые файлы.

Файловый менеджер позволяет выполнять наиболее частые операции - копирования, переноса, удаления, редактирования текстовых файлов, гибкого запуска программ для работы с этими файлами…

Помимо основных функций, многие файловые менеджеры включают ряд дополнительных возможностей, например, таких как работа с сетью (через FTP, NFS и т. п.), резервное копирование, управление принтерами и пр.

Существует два вида файловых менеджеров — навигационные и ортодоксальные. Основное отличие — в последних имеется две панели, реализована соответствующая модель работы.

Целью данной курсовой работы является создание программного продукта «Файл-менеджер». Он в свою очередь сможет облегчить работу с файлами и папками. Для создания продукта использовался язык программирования C#, среда разработки Microsoft Visual Studio 2012.

# 1.Теоретическое введение

## 1.1 Объектно-ориентированное программирование

Истоки ООП начинаются с 60-х годов XX века. Однако окончательное оформление и популяризацию можно отнести к 80-м годам XX века. Особую роль сыграл Алан Кей, сформулировавший основные принципы ООП. В настоящее время большинство проектов реализуются в стиле ООП. Хотя в программировании операционных систем (системном программировании) большую роль играет язык С.

Итак, что же такое ООП? Судя по названию ключевую роль в этой парадигме играет некий объект, а точнее множество объектов. Реальный мир состоит из объектов и их взаимодействий между собой. В результате взаимодействий объекты могут изменяться сами или изменять другие объекты. Поэтому, можно сказать, что ООП является более естественным в каком-то смысле.

Следует понимать, существенную разницу между программой написанной с структурном «стиле» и программой в «стиле» ООП. В первом случае, на первый план выходит логика, понимание последовательности выполнения выражений (действий) для достижения целей. Во-втором — важно системное мышление, умение видеть систему в целом, с одной стороны, и понимание роли ее частей (объектов), с другой.

В объектно-ориентированной технологии используется особый подход к разработке программ, основанный на использовании объектных моделей и нескольких базовых концепциях. К этим концепциям относятся абстрагирование, инкапсуляция, полиморфизм, наследование.

## 1.2. Среда Visual Studio.NET

В 2002 году компания Microsoft создала новую платформу разработки и выполнения программ, которая получила название .NET Framework. Это полностью объектно-ориентированная платформа, которая позволяет использовать уже имеющиеся и создавать собственные типы данных. В . NET под термином “тип” понимаются: классы, структуры, перечисления и иные формы данных. Платформа .NET позволяет разрабатывать компоненты (называемые сборками), которые предоставляют доступ к описанным в них типам другим компонентам (возможно написанным на других языках). Основными целями платформы .NET являлось создание:

• нового формата выполняемых программных модулей – компонент (EXE и DLL), называемых сборками (assembly) или управляемыми модулями, основными особенностями которых является использование общего (независимого от исходного языка) промежуточного языка программирования и метаданных, описывающих все открытые типы данных, содержащиеся в них;

• специальной виртуальной машины (общеязыковой исполняющей среды, common language runtime, CLR), которая управляет компиляцией в инструкции процессора и выполнением модулей, составленных на промежуточном языке; CLR начинает работать при каждом запуске управляемых модулей на выполнение;

• общей библиотеки классов .NET Framework (Framework Class Library, FCL), которые помогают выполнить все базовую функциональность управляемых приложений (например, работа с коллекциями, файлами, сетями, графическим интерфейсом и т.п.).

• набора программных средств, помогающих разрабатывать управляемые модули, например, таких как компиляторы и отладчики; основным средством разработки является интегрированная среда разработки – Visual Studio, позволяющая автоматизировать разработку приложений на всех языках поддерживаемых платформой. Платформа .NET активно развивается и с 2010 года уже используется версия .NET Framework 4.0. Основой .NET Framework является общеязыковая среда выполнения программ (CLR). До разработки данной платформы приложения компилировались в инструкции конкретного процессора, выполнялись в 30 процессах операционной системы Windows (ОС) и могли выполнять самостоятельно любые доступные им действия. С использованием плат- формы .NET, все созданные приложения компилируются в команды общего промежуточного языка (common intermediate language, CIL) и исполняются под управлением CLR. В связи с этим, они называются управляемыми приложениями (managed application). Среда CLR является виртуальной машиной, которая расположена поверх ОС (выполняется под управлением ОС и использует все ее возможности) и управляет выполнением приложений, разработанных для платформы .NET.

## 1.3.  C#  WinForm

C#– это объекто-ориентированное программирование. Он был разработан в 2001 году, инженерами под руководством Андерса Хейлсберга в компании Microsoft. На данное время существует 4 версии языка «си шарп».

Название «Си шарп» (от англ. sharp - диез) происходит от музыкальной нотации, где знак диёз, прибавляемый к основному обозначению ноты, означает повышение соответствующего этой ноте звука на полутон. Это аналогично названию языка С++, где «++» обозначает, что переменная должна быть увеличена на 1.

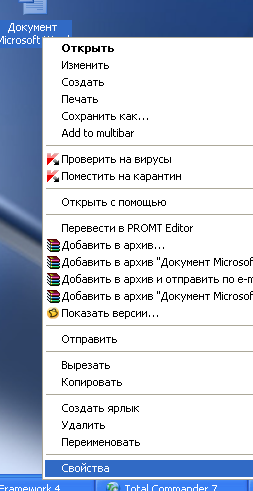
# C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеетстатическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы. Переняв многое от своих предшественников — языков C++, Java, Delphi, Модула и Smalltalk — С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем: так, C# не поддерживает множественное наследование классов (в отличие от C++). C# разрабатывался как язык программирования прикладного уровня для CLR и, как таковой, зависит, прежде всего, от возможностей самой CLR. Это касается, прежде всего, системы типов C#, которая отражает BCL. Так, с развитием CLR от версии 1.1 к 2.0 значительно обогатился и сам C#; подобного взаимодействия следует ожидать и в дальнейшем. (Однако эта закономерность была нарушена с выходом C# 3.0, представляющим собой расширения языка, не опирающиеся на расширения платформы .NET.) CLR предоставляет C#, как и всем другим .NET-ориентированным языкам, многие возможности, которых лишены «классические» языки программирования. Например, сборка мусора не реализована в самом C#, а производится CLR для программ, написанных на C# точно так же, как это делается для программ на VB.NET, J# и др.

# 2. Анализ поставленной задачи и формирование требований

## 2.1 Отношение к виду

1. Разрабатываемый программный продукт относится к системным утилитам. Разрабатываемая программа должна позволять получать основную информацию об открываемом файле, отображать атрибуты, вносить изменения в атрибуты, переименовывать, перемещать и копировать файл.

## 2.2 Аналоги

1. Существует огромное множество подобных программ. Как официальных, так и любительских. Так же компоненты Windows выполняют аналогичные задачи. Далее мы рассмотрим некоторые из них.
2. Свойства папки Windows:
3. Вызываются с помощью нажатия правой кнопки мыши на файл -> свойства. Это наглядно показано на рисунке 1.
4. ****
5. Рисунок 1 – “Вызов”свойств папки Windows
6. После вызова свойств папки, появляется окно, которое отображает описание основных характеристик, просмотр и изменение атрибутов. Вид окна показан на Рисунке 2 и 3

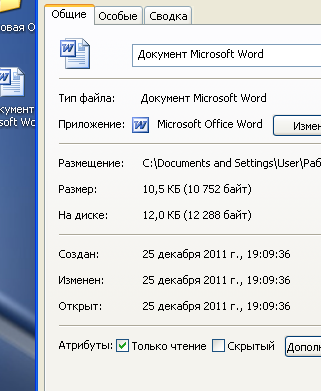
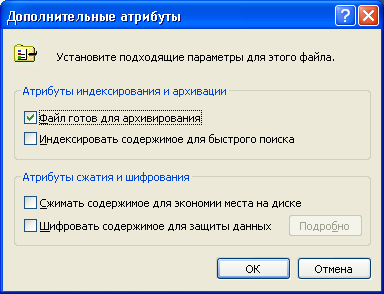
 

Рисунок 2 – Вид окна свойств. Рисунок 3 – Просмотр и изменение атрибутов.

Необходимые функции копирования, переименования и перемещения доступны с помощью других пунктов контекстного меню. Это показано на Рисунке 1

Total Commander

Total Commander (бывший Windows Commander) - это самый мощный и стабильный файловый менеджер для Windows с удобным пользовательским интерфейсом. Total Commander продолжает добрую традицию двухпанельных файловых менеджеров, но отличается повышенным удобством и функциональностью: поддерживает копирование, перемещение, переименование файла, просмотр и изменение его атрибутов, а так же многое другое. К примеру Total Commander позволяет соединятся с FTP серверами и удобно оперировать с содержимым папок узлов сети.

Поддерживает плагины, имеет множество встроенных полезных инструментов.

Интерфейс Total Commander демонстрируется на Рисунке 4.

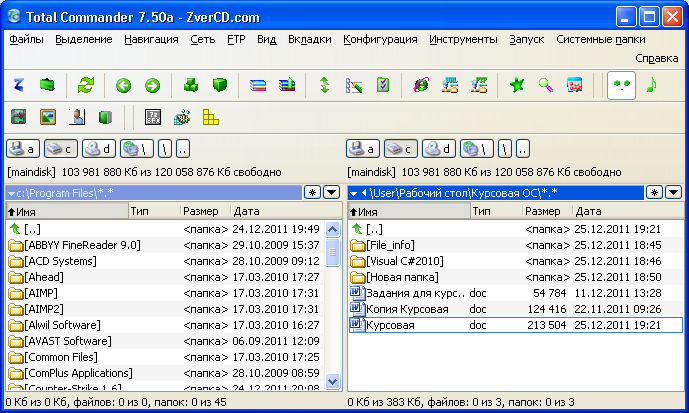
****

Рисунок 4 – Интерфейс Total Commander

Копирование и перемещение выполняются “перетаскиванием” из исходного поля в желаемое.

Недостатком Total Commander может быть невозможность вызова контекстного меню определённо-выбранного файла. Но этот недостаток покрывает состав панели инструментов, с помощью которого можно легко проделать множество операций, в том числе изменение атрибутов. Это наглядно показано на Рисунке 5 и 6.

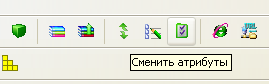
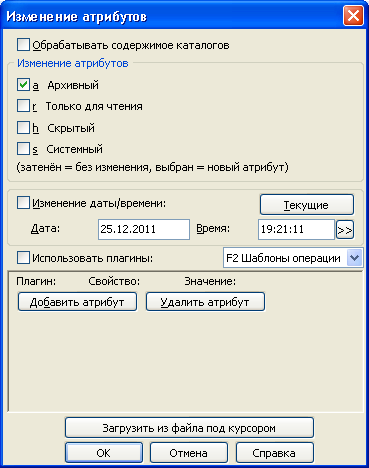
 

Рисунок 5 – Панель инструментов TC. Рисунок 6 – изменение атрибутов файла в TC.

Утилита командной строки Windows

Из командной строки доступны практически все функции, перечисленные выше. Но главным недостатком командной строки – является неудобство. Неопытный пользователь ПК не знает основных команд, поэтому работа с информацией о файле и его атрибутами будет затруднительна. К тому же на введение команд требуется некоторое время, в отличии от TC в котором это проделывается одним нажатием мыши.

Перечислим некоторые команды и наглядно покажем их работу:

* + Команда **attrib** позволяет просматривать и изменять атрибуты файла. Показано на Рисунке 7.

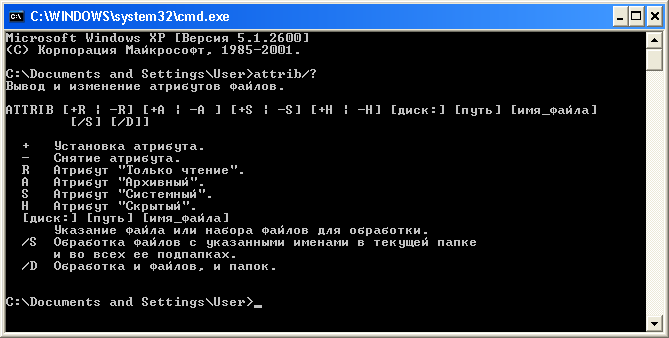


Рисунок 7 – Команда attrib.

Из рисунка видно, что для установки атрибута файла требуется указать полный путь к файлу и его имя, а так же указать буквенное значение, которое соответствует желаемому атрибуту. Это чрезвычайно неудобно.

* команды **copy, moveи rename** (или сокращенный вариант ren) соответственно копируют, перемещают и переименовывают файлы; Синтаксис команды copy показан на Рисунке 8.

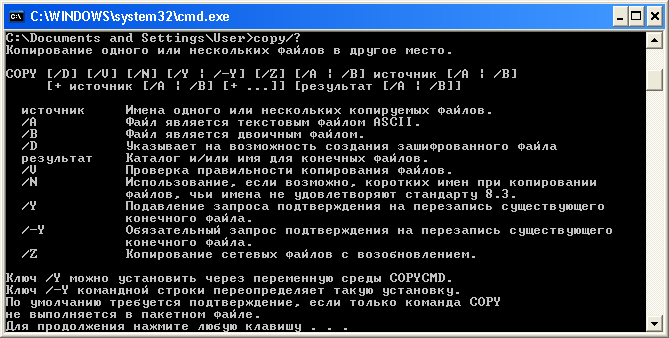


Рисунок 8 – Команда Copy.

Большинство начинающих пользователей ПК даже и не подозревают о половине сказанного в справочнике командной строки.

* команда **dir** позволяет просмотреть характеристики файлов в некоторой папке. Показано на Рисунке 9.

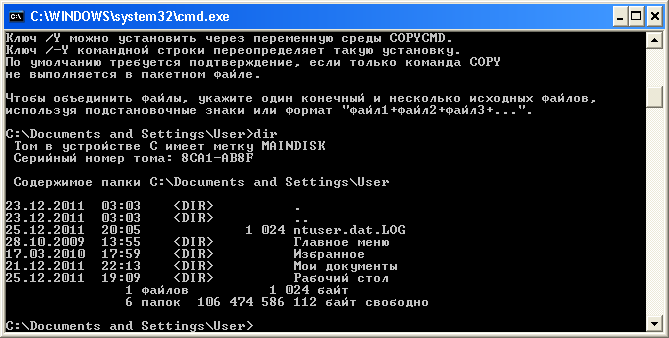


Рисунок 9 – команда dir.

## 2.3 Сравнение

1. На основаниях, рассмотренных в предыдущем пункте проведём сравнение аналогов, указав их преимущества и недостатки
2. Свойства папки Windows

Преимущества:

* Удобный интерфейс
* Простота использования

Недостатки:

* Функции расположены в разных контекстах меню
* Неполный набор атрибутов

Total Commander

Преимущества:

* Удобный интерфейс
* Простота использования
* Широкий набор возможностей
* Польный набор атрибутов
* Возможность добавления персональных атрибутов
* Все функции выполняются в одном окне

Недостатки:

* Невозможность вызова контекстного меню выбранного файла

Утилита командной строки Windows

Преимущества:

* Входит в состав ОС
* Все функции доступны из одного окна

Недостатки:

* Неудобство интерфейса.
* Неудобство использования для начинающего пользователя ПК.
* Долговременность ввода синтаксиса

## 2.4 Требования

Основными требованиями к разрабатываемой программе являются:

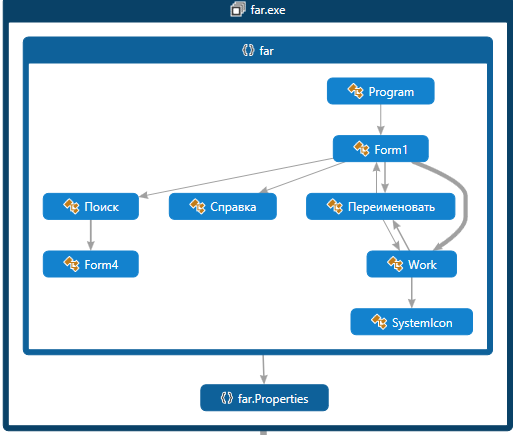
1. 1. Удобство использования
2. 2. Простота интерфейса
3. 3. Выполнение определённы функций:

* Открытие файла
* Копирование
* Перемещение
* Переименование
* Вывод сообщений о удачном перемещении, переименовании, копировании.
* Вывод сообщений о не найденном файле, ошибке и т.д.

# 3.Диаграмма классов

Интерфейс программы должен быть интуитивно понятен, не загроможден лишними деталями. Проект состоит из двадцати одной формы, двух отчетов, файла ресурсов и настроек. Общая схема структуры проекта представлена на схеме 1.

Схема 1 - Структура проекта



Схемам 2 – Диаграмма классов

Диаграмма классов проекта представлена на Схеме 2. Как видно из диаграммы, проект содержит класс Work, который одержит в себе все основный методы, предназначенные для (Схема 3):

* Получения списка запоминающих устройств
* Удаление файлов и папок
* Создание новой папки
* Копирование элементов (рекурсивное копирование)
* Перемещение файлов и папок
* Переименование выделенного объекта

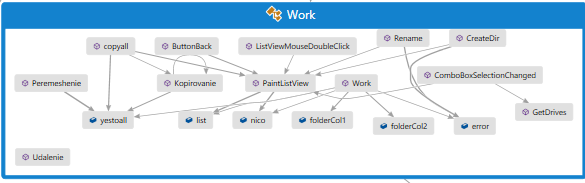


Схема 3 – класс Work

Все остальные классы наследуют его, чтобы получить данный функционал.

Класс Form1 содержит в себе различные обработчики событий:

* Обработчик двойного клик мыши по объектам
* Обработчик одиночного клик мыши по объектам
* Обработчик выбора диска
* Обработчики кнопок (удалить, копировать, переименовать, назад и т. д.)
* Обработка нажатия пункта меню

Так же обработчики содержат в себе различные проверки на ошибки: выбран элемент или нет, активна панель или нет, различные ошибки при копирование, перемещении и т. д. (Схема 4)

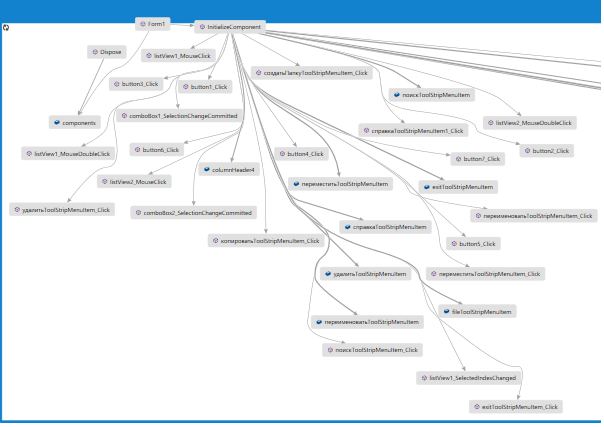


Схема 4 – класс Form1

Класс Form4 предназначен для реализации поиска. Он содержит в себе класс Searcher. Данный класс соответствует объекту-потока поиска. Атрибуты класса содержат приватные переменные, которые соответствуют параметрам поиска. Класс содержит в себе метод StartSearch – реализует поиск во всех ветвях и подкаталогах директории заданное как Searcher.parentDirectory. И еще один метод SearchFile – реализует поиск за заданными критериями отбора в каталоге, который передается как параметр метода. (Схема 5)

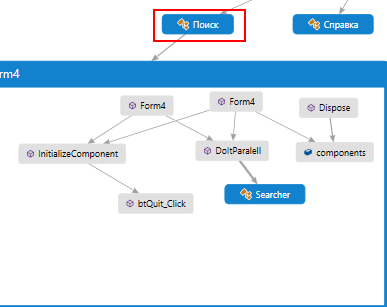


Схема 5 – класс Form 4

Класс SystemIcon предназначен для реализации иконок в приложении.(Схема 6)

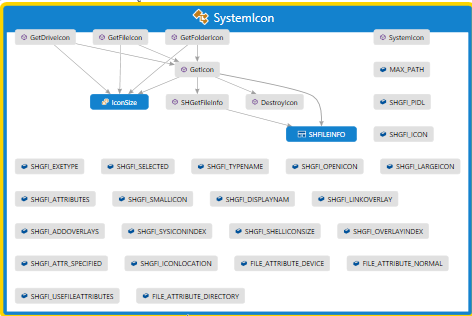


Схема 6 – класс SystemIcon

Класс DirectoryInfo предназначен для работы с папками. Он позволяет делать стандартные операции - перемещение, удаление, создание папок и подпапок и др.. Кроме того, методы этого класса позволяют получить все подкаталоги и файлы указанной папки. В отличие от класса Directory, методы класса DirectoryInfo - это методы экземпляра (т. е. они не статические).

Класс Thread - представляющий отдельный поток. Он определяет ряд методов и свойств, которые позволяют управлять потоком и получать информацию о нем. Используются такой метод как Start - запускает поток.

Класс FileInfo позволяет получать подробности относительно существующих файлов на жестком диске (т.е. время создания, размер и атрибуты) и предназначен для создания, копирования, перемещения и удаления файлов.

Класс Directory предоставляет статические методы для создания, перемещения и перечисления в каталогах и вложенных каталогах. Этот класс не наследуется.

# 4.Руководство пользователя

* + 1. Описание ВИ выбора диска для панели:

ВИ: Выбрать диск

Краткое описание: из предложенного списка выбирается диск, как для первой, так и для второй панели.

Действующие лица: Пользователь

Предусловия:

1. Пользователь должен запустить программу.
2. Пользователь должен определиться с необходимым ему диском.

Основной поток:

1. Программа предоставляет пользователю выбрать диск.
2. Программа отображает содержимое диска со всеми папками и файлами в панели. (рисунок 10)

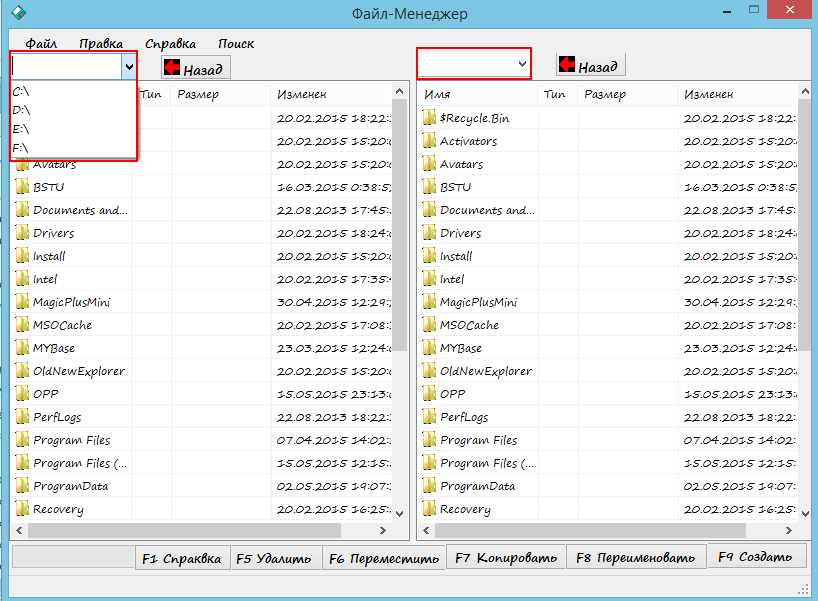
1

Рисунок 10 – выбор системного диска

2.Описание расширяющего ВИ «Копировать»:

ВИ: Копировать файл

Краткое описание: Программа копирует файл из одной панели в другую

Действующие лица: Пользователь

Предусловия:

1. Пользователь должен запустить программу.

2. Пользователь должен выбрать желаемый файл в одной панели.

3. Пользователь должен выбрать в другой панели место (папку) куда хочет копировать файл.

4. Пользователь должен нажать кнопку «Копирование».

5. Появляется окно, в котором пользователю предлагается нажать «Ок» для продолжения. (Рисунок 11)

В указанном месте появляется копия файла либо папки.

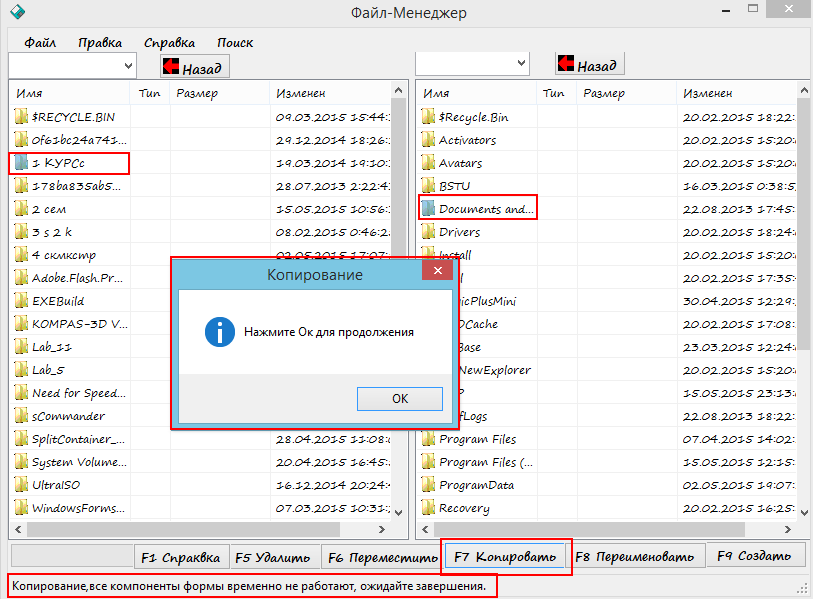


Рисунок 11 – Копирование файла

1. Описание ВИ Переименовать:

ВИ: Переименование файла/папки

Краткое описание: Выполняется изменение имени файла или папки.

Действующие лица: Пользователь

Предусловия:

1. Пользователь должен запустить программу.

2. Пользователь должен выбрать желаемый файл/папку.

3. Пользователь должен нажать кнопку «Переименовать».

4. Пользователь должен указать желаемое имя файла/папки. (Рисунок 12)

У файла изменяется имя на указанное.

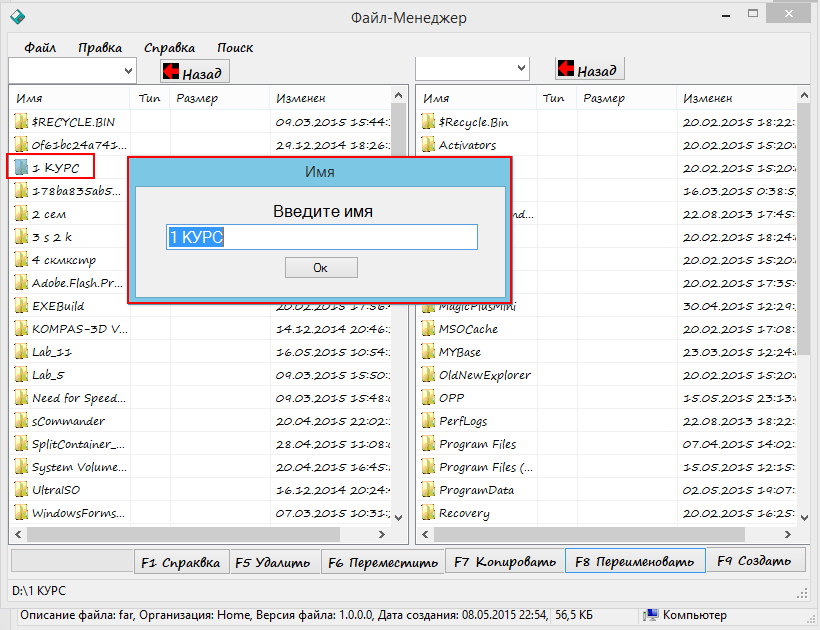


Рисунок 12- Переименовать файл

1. Описание расширяющего ВИ «Переместить»:

ВИ: Переместить файл

Краткое описание: Программа перемещает файл из одной панели в другую

Действующие лица: Пользователь

Предусловия:

1.Пользователь должен запустить программу.

2. Пользователь должен выбрать желаемый файл в одной панели.

3. Пользователь должен выбрать в другой панели место (папку) куда хочет переместить файл.

4. Пользователь должен нажать кнопку «Переместить».

5. Появляется окно, в котором пользователю предлагается нажать «Ок» для продолжения. (Рисунок 13)

В указанном месте появляется файл.

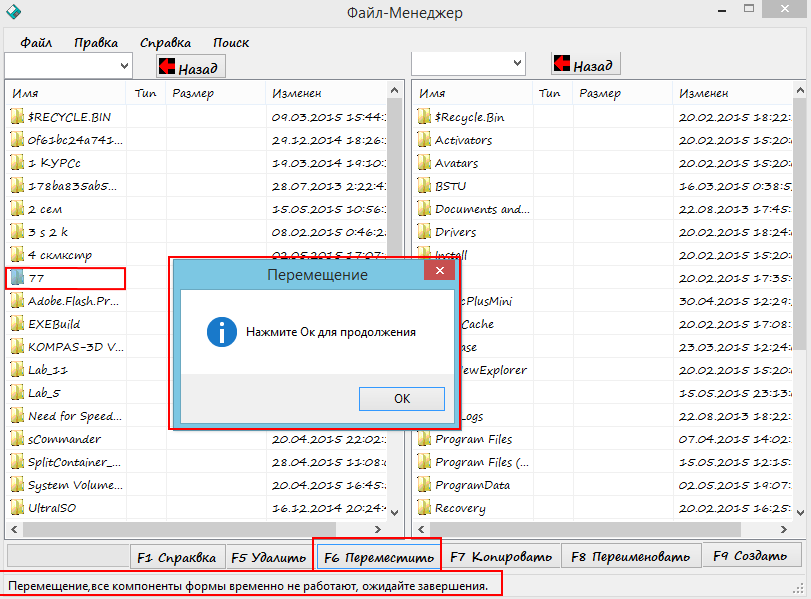


Рисунок 13 – Перемещение файла/папки

5.Описание расширяющего ВИ «Удалить»:

ВИ: Удалить файл/папку

Краткое описание: Программа удаляет файл из одной из панелей

Действующие лица: Пользователь

Предусловия:

1. Пользователь должен запустить программу.
2. Пользователь должен выбрать желаемый файл/папку в одной из панелей.
3. Пользователь должен нажать кнопку «Удалить».
4. Появляется окно, в котором у пользователя спрашивается «Вы уверены что хотите удалить выбранные элементы?» Пользователь выбирает «Да» или «Нет». (Рисунок 14)

Из указанного места исчезает файл/папка.

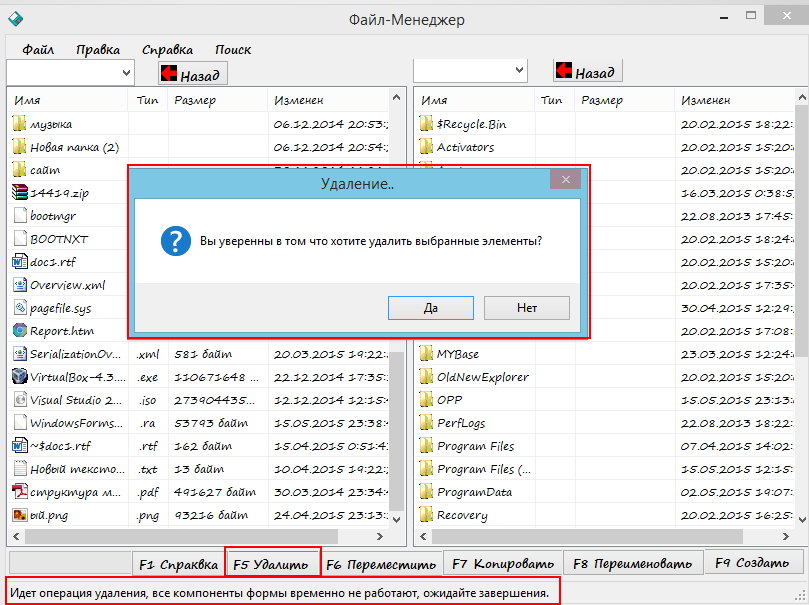


Рисунок 14 - Удаление файла/папки

6.Описание расширяющего ВИ «Поиск»:

ВИ: Найти файл/папку

Краткое описание: Программа файлы и папки на выбранном диске

Действующие лица: Пользователь

Предусловия:

1. Пользователь должен выбрать в меню «Поиск».
2. Пользователь должен выбрать в появившемся окне диск или другое расположение, в котором он хочет осуществить поиск.
3. Пользователь должен выбрать по чем будет производиться поиск:

а) Имя;

б) Дата создания;

в) Размер;

1. После того как пользователь сделал свой выбор он должен нажать кнопку «Поиск». (Рисунок 15-16)

Появится окно с результатом поиска.

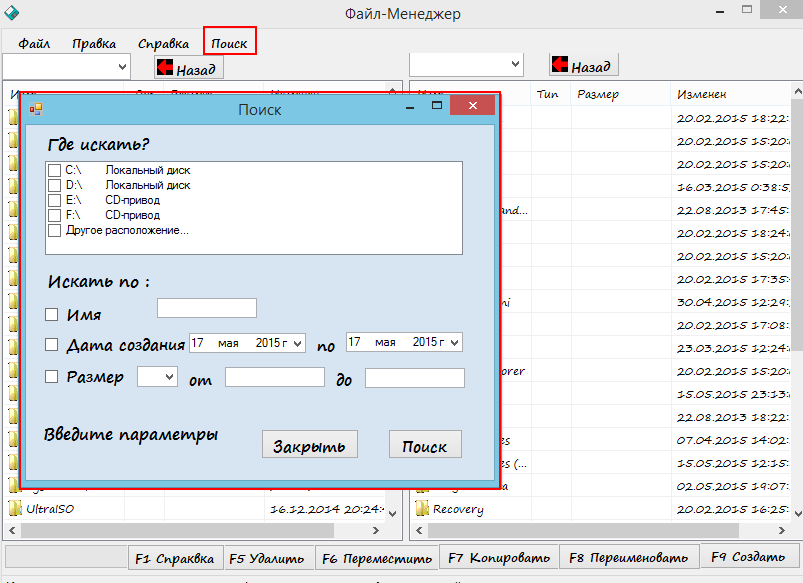


Рисунок 15 – Окно поиска

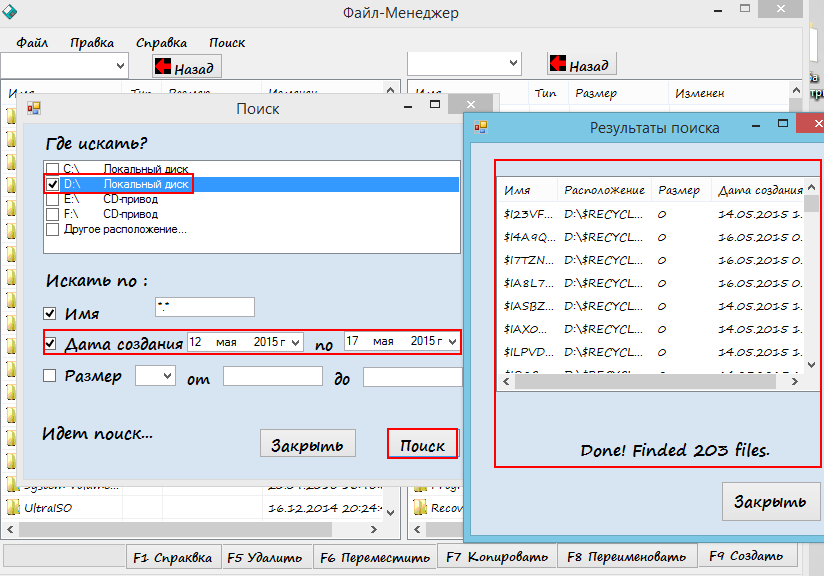


Рисунок 16 – Окно с результатом поиска

Так же пользователь может воспользоваться пунктом меню «Справка», «О программе» (Рисунок 17)

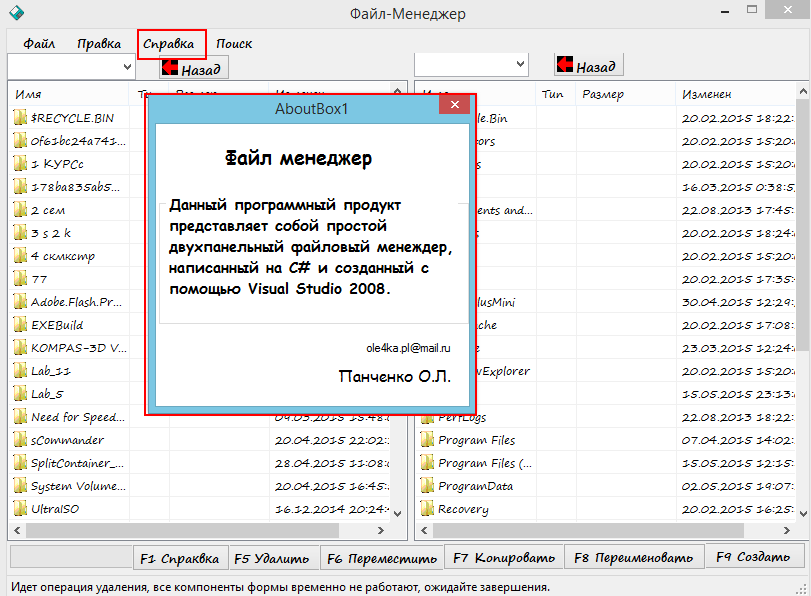


Рисунок 17 – Окно «О программе»

Также пользователь может закрыть приложения либо нажав на кнопку крестик либо же через пункт меню. (Рисунок 18)

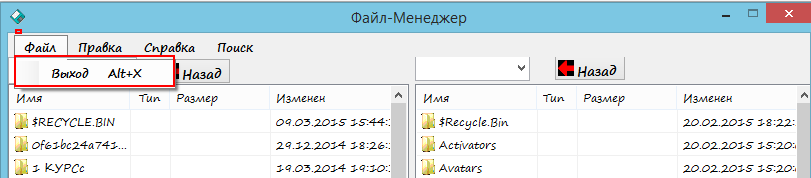
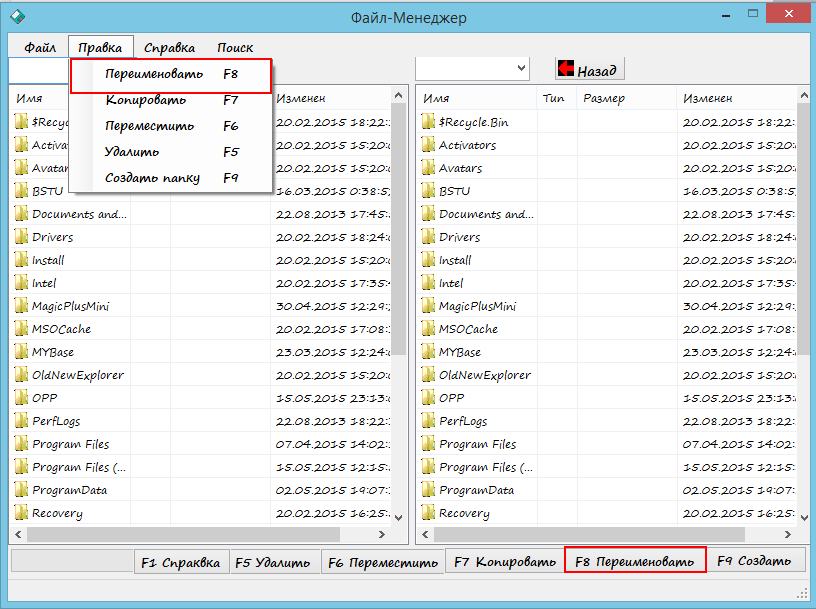


Рисунок 16 – Выход

Также пользователю предоставляется возможность пользовать либо меню, либо кнопками в таких командах как: копировать, переместить, удалить, переименовать, создать, справка. А так эти команды заменяют кнопки клавиатуры. (Рисунок 19)



Копировать – F7;

Переместить – F6;

Переименовать – F8;

Создать – F4;

Удалить – F5;

Справка – F1;

Выход – Alt+X;

Рисунок 19 – Сочетания кнопок

# Описание программы

## Использование пространств имен Net Framework

System – cодержит классы, позволяющие сопоставлять коды URI с шаблонами URI и группами шаблонов URI.

System.Collections.Generic – содержит интерфейсы и классы, определяющие универсальные коллекции, которые позволяют пользователям создавать строго типизированные коллекции, обеспечивающие повышенную производительность и безопасность типов по сравнению с не универсальными строго типизированными коллекциями.

System.Drawing – содержит типы, поддерживающие базовые графические функции GDI+. Дочерние пространства имен поддерживают более сложные функции двухмерной и векторной графики, дополнительные функции обработки изображений, а также службы, связанные с эпиграфикой печатью. Дочернее пространство имен также содержит типы, которые расширяют логические и графические возможности пользовательского интерфейса во время разработки.

System.ComponentModel - содержит классы, реализующие поведение компонентов и элементов управления во время проектирования и выполнения. Данное пространство имен включает базовые классы и интерфейсы, предназначенные для реализации преобразователей атрибутов и типов, для привязки к источникам данных и для лицензирования компонентов.

System.Data - обеспечивает доступ к классам, которые представляют архитектуру ADO.NET. Архитектура ADO.NET позволяет создавать компоненты, эффективно работающие с данными из различных источников.

System.Text – содержит типы для работы с кодировками символов и для управления строками. Дочернее пространство имен позволяет обрабатывать текст с использованием регулярных выражений.

System.Linq - содержит классы и интерфейсы, которые поддерживают запросы, использующие LINQ (Language-Integrated Query).

System.Windows.Forms – содержит классы для создания приложений Windows, которые позволяют наиболее эффективно использовать расширенные возможности пользовательского интерфейса, доступные в операционной системе Microsoft Windows.

System.IO – содержит типы, поддерживающие ввод и вывод, включая возможности чтения и записи данных в потоках, как синхронно, так и асинхронно, сжатия данных в потоках, создания и использования изолированных хранилищ, сопоставления файлов логическим адресным пространствам приложений, хранения различных объектов данных в одном контейнере, взаимодействия с использованием анонимных или именованных каналов, реализации пользовательского ведения журнала и обработки входящих и исходящих потоков данных в последовательных портах.

System.Threading - содержит классы и интерфейсы, которые дают возможность программировать в многопоточном режиме. В дополнение к классам, чтобы синхронизировать действия потока и доступ к данным. Данное пространство имен включает класс [ThreadPool](https://msdn.microsoft.com/ru-ru/system.threading.threadpool), который позволяет использовать кластера система- предоставленных потоков и класс [Timer](https://msdn.microsoft.com/ru-ru/system.threading.timer), который выполняет обратные вызовы на потоках из пула потоков.

System.Runtime.InteropServices **-** предоставляет широкий спектр членов, поддерживающих COM-взаимодействие и службы вызова неуправляемого кода.

## 5.2 Реализуемые функции

**Вывод списка файлов и папок (класс Worck)**

Для начала нужно знать путь (диск) к своему текущему местоположению, эту информацию следует хранить отдельной строкой (у нас это будет переменная **Tag**, Все манипуляции осуществляются с файлами и папками относительно этого пути. Также имеются две переменные,которые служат для хранения историй папок для каждой панели.

// для хранения историй папок для каждой панели

public System.Collections.Specialized.StringCollection folderCol1;

public System.Collections.Specialized.StringCollection folderCol2;

Функция для получения запоминающих устройств в системе

public void GetDrives(ComboBox comb)

{try

{string[] drivesArray = Directory.GetLogicalDrives();/ /Объявление массива строк и заполнение его именами устройств

//Заполнение комбобоксов (первого и второго) именами устройств

foreach (string s in drivesArray)

{ comb.BeginUpdate();

comb.Items.Add(s);

comb.EndUpdate(); } }

catch{MessageBox.Show("Устройство не готово", "Error!", MessageBoxButtons.RetryCancel, MessageBoxIcon.Error); }}

Заполнение ListView именами каталогов и файлов

public void PaintListView(ListView lvs, string putb)

{ try

{ ListViewItem lvi;// элемент проссматриваегомого списка

ListViewItem.ListViewSubItem lvsi;// субэлементы (столбец списка)

lvs.SmallImageList = list;

lvs.LargeImageList = list;

lvs.BeginUpdate();

lvs.Items.Clear();

// получение инф. о папке

System.IO.DirectoryInfo dir = new System.IO.DirectoryInfo(putb);

// извлечение файлов и папок, находящихся в папке

System.IO.DirectoryInfo[] dirs = dir.GetDirectories(); // папки

System.IO.FileInfo[] files = dir.GetFiles(); // файлы

// проход в цикле по всем ПАПКАМ и включение их в список

foreach (DirectoryInfo di in dirs)

{ // создание основного элемента ListViewItem

Icon ic = SystemIcon.GetFolderIcon(putb + di.Name, SystemIcon.IconSize.Small, false); list.Images.Add(ic);

lvi = new ListViewItem();

lvi.Text = di.Name; // имя папки

lvi.ImageIndex = nico++; // индекс иконки (папки)

lvi.Tag = di.FullName; // полный путь к папке

// создание 2 элементов ListVIewSubItem (для 2 доп.столбцов)

lvsi = new ListViewItem.ListViewSubItem();

lvsi.Text = ""; // у папок размер отсутствует

lvi.SubItems.Add(lvsi); // включение элемента в ListViewItem

lvsi = new ListViewItem.ListViewSubItem();

lvsi.Text = ""; // последний столбец

lvi.SubItems.Add(lvsi); // включение элемента в ListViewItem

lvsi = new ListViewItem.ListViewSubItem();

lvsi.Text = di.LastAccessTime.ToString(); // последний столбец

lvi.SubItems.Add(lvsi); // включение элемента в ListViewItem

// включение элемента ListViewItems в семейство Items

lvs.Items.Add(lvi); }

// проход в цикле по всем ФАЙЛАМ и включение их в список

foreach (FileInfo fi in files)

{ // создание основного элемента ListViewItem

lvi = new ListViewItem();

Icon ic = SystemIcon.GetFileIcon(putb + fi.Name, SystemIcon.IconSize.Small, false);

list.Images.Add(ic);

lvi.Text = fi.Name; // имя файла

lvi.ImageIndex = nico++; // индекс иконки (папки)

lvi.Tag = fi.FullName; // полный путь к файлу

// создание 2 элементов ListVIewSubItem (для 2 доп.столбцов)

lvsi = new ListViewItem.ListViewSubItem();

lvsi.Text = fi.Extension.ToString(); // у папок размер отсутствует

lvi.SubItems.Add(lvsi); // включение элемента в ListViewItem

lvsi = new ListViewItem.ListViewSubItem();

lvsi.Text = Convert.ToString(fi.Length.ToString() + " " + "байт"); // последний столбец lvi.SubItems.Add(lvsi); // включение элемента в ListViewItem

lvsi = new ListViewItem.ListViewSubItem();

lvsi.Text = fi.LastAccessTime.ToString(); // последний столбец

lvi.SubItems.Add(lvsi); // включение элемента в ListViewItem

// включение элемента ListViewItems в семейство Items

lvs.Items.Add(lvi); }

lvs.EndUpdate();//Финализация обновления listview

} catch (Exception err)

{MessageBox.Show("Ошибка:" + err.Message + " Выберите в списе другое устройство.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

lvs.EndUpdate();

return; }}

**Операции с файлами и каталогами.**

Файловый менеджер обязан уметь выполнять различные действия над файлами. Но прежде чем что сделать с файлом или каталогом нужно получить его имя и путь до него. Для этого необходимо отслеживать выделенный элемент из ListViewи знать является ли он файлом или каталогом.

Для копирования папки в место назначения используется рекурсивная ыункция копирования Kopirovanie:

public void Kopirovanie(DirectoryInfo source, DirectoryInfo target)

{try

{ // Если директория для копирования файлов не существует, то создаем ее

if (Directory.Exists(target.FullName) == false)

{ Directory.CreateDirectory(target.FullName);}

// Копируем все файлы в новую директорию

foreach (FileInfo fi in source.GetFiles())

{ if (!File.Exists(Path.Combine(target.ToString(), fi.Name)))//Еслы файл с таким именем не существует в месте назначения - то окпируем

{fi.CopyTo(Path.Combine(target.ToString(), fi.Name), true);}

else//Если в месте назначения уже существует файл с таким именем

{ if (yestoall == false)

{ DialogResult result;

//Запрос к пользователю на разрешение замещения файла

string.Format("В узанном месте уже существует файл с таким именем ({0}). Заменить?", Path.Combine(target.ToString(), fi.Name));

result = MessageBox.Show(s, "Копирование", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

//Если да

if (result == System.Windows.Forms.DialogResult.Yes)

{ //Запрос на замещение последующих файлов без обращения к пользователю result = MessageBox.Show("Заменять все последующие файлы?", "Копирование", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

//Если пользователь подтвердит - устанавливаем флаг yestoall

if (result == System.Windows.Forms.DialogResult.Yes) yestoall = true;

//Удаляем существующий файл и замещаем его на файл из источника

File.Delete(Path.Combine(target.ToString(), fi.Name));

fi.CopyTo(Path.Combine(target.ToString(), fi.Name), true); }}

//Если флаг yestoall установлен - замещаем файл без разрешения

else { File.Delete(Path.Combine(target.ToString(), fi.Name));

fi.CopyTo(Path.Combine(target.ToString(), fi.Name), true); }}}

// Копируем рекурсивно все поддиректории

foreach (DirectoryInfo diSourceSubDir in source.GetDirectories())

{ // Создаем новую поддиректорию в директории

DirectoryInfo nextTargetSubDir =

target.CreateSubdirectory(diSourceSubDir.Name);

// Опять вызываем функцию копирования

// Рекурсия

Kopirovanie(diSourceSubDir, nextTargetSubDir);}

catch { MessageBox.Show("Не удается совершить операцию копирования", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;}}

public void copyall(ListView sourse, ListView destination,ref System.Collections.Specialized.StringCollection folder1,ref System.Collections.Specialized.StringCollection folder2,ToolStripStatusLabel tssl,ProgressBar pb)

{try

{ yestoall = false;

//Вывод в строку состояния информации о том что идет операция копирования

tssl.Text = "Копирование,все компоненты формы временно не работают, ожидайте завершения.";

//Вывод информационного сообщения о том, что идет копирование. Оно нужно для того чтобы отобразился результат

//операции которая выполнилась выше. Если этого сообщения не будет, пользователь не увидит сообщения в строке состояния

MessageBox.Show("Нажмите Ок для продолжения", "Копирование", MessageBoxButtons.OK, MessageBoxIcon.Information);

//Инициализация progressbar

pb.Minimum = 1;

// Установить Максимальное общее количество файлов для копирования.

pb.Maximum = sourse.SelectedItems.Count;

// Задать начальное значение элемента управления progressbar.

pb.Value = 1;

pb.Step = 1;

//Цикл поочередного копирования выделенных папок и файлов

for (int j = 0; j <= sourse.SelectedItems.Count - 1; j++)

{string filename1 = sourse.SelectedItems[j].Tag.ToString();

//Строка которая содержить путь к j-ому файлу или папке

//Строка, созданная обьединением пути назначения и имени j-го элемента string filename2 = folder2[folder2.Count - 1] + "\\" + sourse.SelectedItems[j].Text.ToString();

if (sourse.SelectedItems[j].ImageIndex == 0)//Если папка

{if (!Directory.Exists(filename2))

//Если в месте назначения папка с таким названием не существует

{ Directory.CreateDirectory(filename2); }//Создаем папку

folder2.Add(filename2);//Добавляем путь этой папки в историю приемника

//Создания списков информации о файлах и папках, которые находятся в в копируемой папке

DirectoryInfo dir1 = new DirectoryInfo(sourse.SelectedItems[j].Tag.ToString());

DirectoryInfo dir2 = new DirectoryInfo(folder2[folder2.Count - 1]); Kopirovanie(dir1, dir2);

//Вызов функции, которая рекурсивно копирует в приемник все папки и файлы из источника

folder2.RemoveAt(folder2.Count - 1); }//Выход из папки которую только что скопировали

else//Если файл

{if (!File.Exists(filename2))

//Если в приемнике нет файла с таким же именем - копируем файл

{ File.Copy(filename1, filename2);}

else//Иначе

{ if (yestoall == false)

//Если флаг yestoall не установлен - спрашиваем о замене файла у пользователя { DialogResult result;

string s = string.Format("В узанном месте уже существует файл с таким именем:{0}. Заменить?", filename1);

result = MessageBox.Show(s, "Копирование", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

//Если подтверждено - спрашиваем о замене всех последующих

if (result == System.Windows.Forms.DialogResult.Yes)

{ result = MessageBox.Show("Заменять все последующие файлы?", "Копирование", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

//Если подтверждено - то все последующие одинаковые файлы будут заменены без спроса

if (result == System.Windows.Forms.DialogResult.Yes) yestoall = true;

File.Copy(filename1, filename2, true); }//Замещение файла в приемнике else continue; }

//Если пользователь отменил замещение - пропускаем этот файл и переходим к следующему элементу

else File.Copy(filename1, filename2, true); }}

//Флаг yestoall установлен, поэтому замещаем файл в приемнике без разрешения

pb.PerformStep();}//Увеличение содержимого progressbar на 1 шаг

pb.Value = 1;//Обнуление progressbar

tssl.Text = "Готово";

//Вывод в строку состояния информации об успешном выполнении операции

//Обновление списков файлов и папок на обеих панелях

PaintListView(sourse, folder1[folder1.Count - 1]);

PaintListView(destination, folder2[folder2.Count - 1]); }

catch

{MessageBox.Show("Не удается совершить операцию копирования", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return; }}

Функция поиска

Для поиска в классе Form4 используется метод, который реализует многопоточность, создает коллекции классов Searcher и Thread (папок), передает в патоки делегаты метода Searcher.StartSearch, который реализует поиск файлов по введенным критериям.

private void DoItParalell(DirectoryInfo di)

{ //если в директории есть вложенные папки выполнится следующий код

if (di.GetDirectories().Length != 0)

{ //создадим новый список екземпляров поискового класса

Searchers = new List<Searcher>();

//создадим новый список потоков

Threads = new List<Thread>();

//для каждой папки в корневой выполним создание поискового класса и потока

//для передачи ему делегата на метод поиска

foreach (DirectoryInfo subdir in di.GetDirectories())

{Searcher s1 = new Searcher(subdir, this.FileName, this.BegDate, this.EndDate, this.MinSize, this.MaxSize);

this.Searchers.Add(s1);

Thread t1 = new Thread(new ThreadStart(s1.StartSearch));

this.Threads.Add(t1); }

//запускаем потоки на выполнение

foreach (Thread item in this.Threads)

{ item.Start();}

//ждем завершения работы всех потоков

while (true)

{int endCount = 0;

foreach (Thread item in this.Threads)

{ //проверка состояния потока

if (item.ThreadState == ThreadState.Running)

{ endCount++;} }

//если вторичных потоков больше нет основной поток продолжает работу

//прерывая текущий цикл

if (endCount == 0) break;}

int counter = 0;

//Для каждого поискового класса читаем список найденных файлов

foreach (Searcher si in Searchers)

{ //Добавляем информацию про каждый полученный файл в список на форме

foreach (FileInfo fi in si.FindedFiles)

{ this.listView1.Items.Add(fi.Name);

this.listView1.Items[counter].SubItems.Add(fi.DirectoryName);

this.listView1.Items[counter].SubItems.Add((fi.Length / 1000).ToString());

this.listView1.Items[counter].SubItems.Add(fi.CreationTime.ToString());

counter++; }}

//Создаем екземпляр поискового класса для поиска в корен дисков/входных директорий

Searcher ls = new Searcher(di, this.FileName, this.BegDate,this.EndDate,this.MinSize, this.MaxSize);

//Ищем файлы только внутри корневой директории

List<FileInfo> localsFile = ls.SearchFile(di);

//Добавляем найденные файлы в визуальный список на форме

foreach (FileInfo fi in localsFile)

{ this.listView1.Items.Add(fi.Name);

this.listView1.Items[counter].SubItems.Add(fi.DirectoryName);

this.listView1.Items[counter].SubItems.Add((fi.Length / 1000).ToString());

this.listView1.Items[counter].SubItems.Add(fi.CreationTime.ToString());

counter++; }

//Выводим сообщение о количестве отысканных файлов

this.lbProgress.Text = "Done!" + " Finded " + listView1.Items.Count + " files.";

//Если файлов в поисковых классах не обнаружено в метку на форме

//выводится сообщение "File not found"

if (counter == 0)

{ this.lbProgress.Text = "File not found"; }}

//этот код выполняется если в корневом каталоге нет вложенных

else

{здается единый екземпляр класс Searcher

//Его аттрибуты инициализируются привтаными переменными текущей формы

Searcher singleSearch =new Searcher(di this.FileName,this.BegDate,this.EndDate,this.MinSize, this.MaxSize);

//Проводится поиск в одном потоке

singleSearch.StartSearch();

int counter = 0;

//Получаем все отысканные файлы и выводим в список на форме

foreach (FileInfo fi in singleSearch.FindedFiles)

{ this.listView1.Items.Add(fi.Name);

this.listView1.Items[counter].SubItems.Add(fi.DirectoryName);

this.listView1.Items[counter].SubItems.Add((fi.Length / 1000).ToString());

this.listView1.Items[counter].SubItems.Add(fi.CreationTime.ToString());

counter++;

this.lbProgress.Text = "Готово!"; }

//Если файлы не найдены - выводим сообщение

if (counter == 0)

{this.lbProgress.Text = "Файл не найден";} } }

Метод, который ищет все файлы в переданной в виде параметра директории что отвечает критериям поиска

public List<FileInfo> SearchFile(DirectoryInfo dir)

{ string pattern = this.fileName;

List<FileInfo> SearchedFile = new List<FileInfo>();

//Отбираем файлы которые соответствуют строке поиска

FileInfo[] inDest = dir.GetFiles(pattern, SearchOption.TopDirectoryOnly);

foreach (FileInfo fi in inDest)

{ //Попытка доступа к файлам, если успешно - сравниваем даты и размеры с эталонными

try {

if (fi.CreationTime.Date >= this.crTimeBeg && fi.CreationTime.Date <= this.crTimeEnd && ((fi.Length) / 1000) <= this.sizeInKbEnd && ((fi.Length / 1000)) >= this.sizeInKbBegin)

{ try { SearchedFile.Add(fi); }

catch { Thread.Sleep(1); }} }

catch { Thread.Sleep(1); } }

//Возвращаем список файлов

return SearchedFile; }

# 6.Тестирование

Для оценки работоспособности разрабатываемого приложения проводится его тестирование. В ходе работы с приложением пользователь может допускать ряд ошибок, на который программа должна как-то реагировать и сообщать пользователю о неправильности его действий. Для выполнения тестирования необходимо проработать все возможные варианты действий пользователя.

В случае выбора пользователем системного устройства, которого не существует появляется сообщение об ошибке «Устройство не готово. Выберете другое устройство.». Пользователь нажимает «ОК». Панель остается пустой. (Рисунок 20)

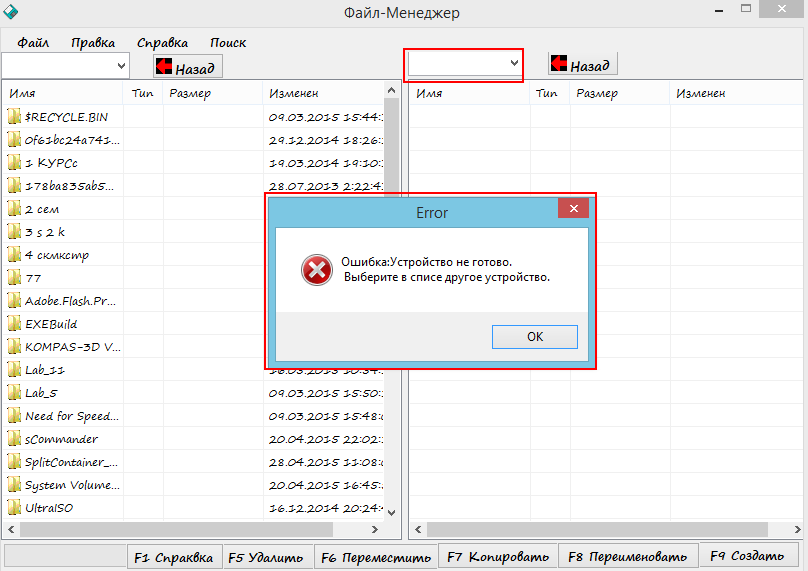


Рисунок 20 – Неправильный выбор системного устройства

В случае, когда пользователь пытается копировать объект из одной панели в другую, то если объект с таким именем уже существует – выводиться сообщение о том, что такой файл уже существует и пользователю может заменить его. Пользователь выбирает «Да» или «Нет». (Рисунок 21)

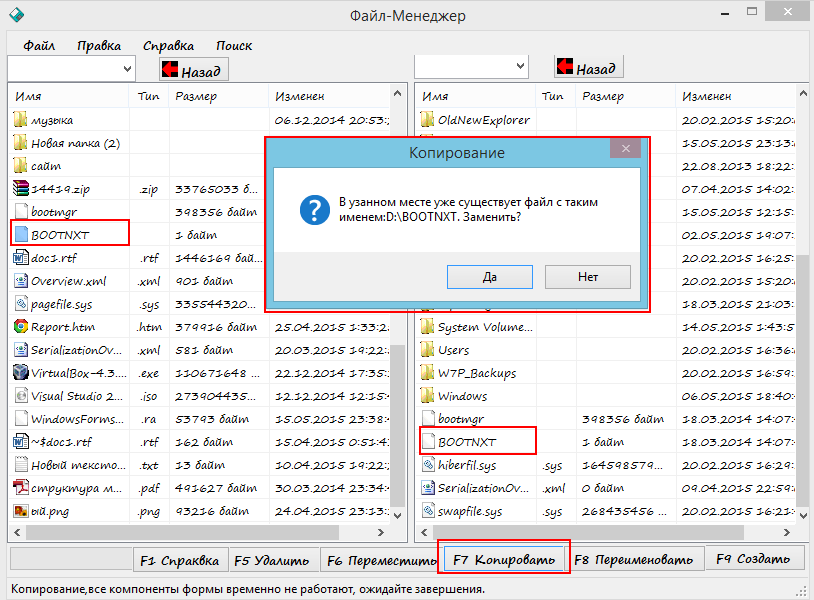


Рисунок 21 – Попытка скопировать уже имеющийся файл

Случай, когда пользователь в одной из панелей не выбирает никакого системного устройства, но пытается туда что-то скопировать, то появляется сообщение о невозможности такой операции. (Рисунок 22). Такое же сообщение выводиться при попытки пользователя выполнить эти действия для перемещения файла или папки (Рисунок 23)

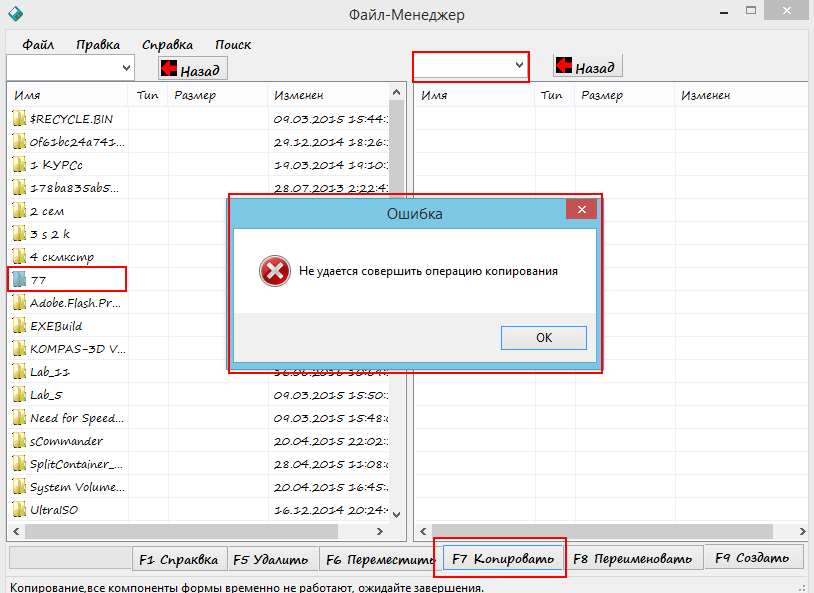


Рисунок 22 – Попытка копирования не удалась

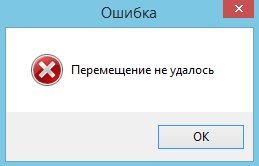
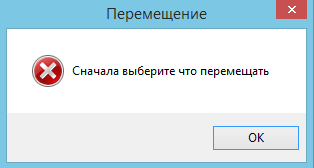
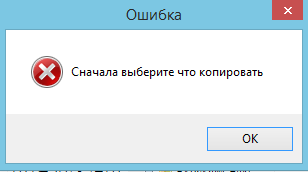


Рисунок 23 – Попытка переместить файл не удалась

Так же при случае если пользователь не выбрал никаких объектов, но пытается копировать, удалить, переименовать то выводится сообщение о просьбе выбрать объекты (Рисунок 24)

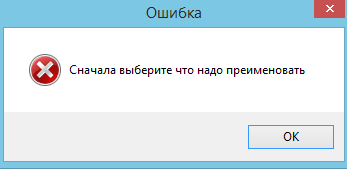
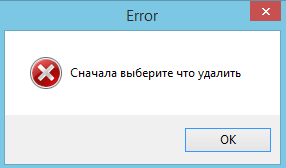
 

Рисунок 24 – Не выбраны объекты

Если пользователь при попытки переименовать файл или папку оставляет поле не заполненным, то программа выведет сообщение о том, что поле не должно быть пустым, и пользователь должен его обязательно заполнить. (Рисунок 25)

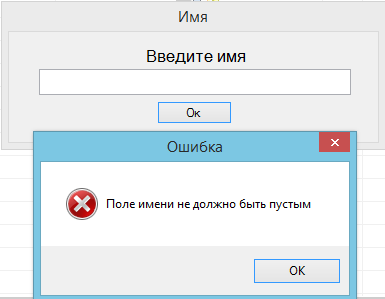


Рисунок 25 – Незаполненное поле

Если пользователь не изменяет имя, а оставляет его таким же, то программа выдает сообщение о том, что «Переименовать не удалось». (Рисунок 26)

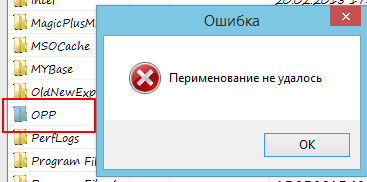


Рисунок 26 – имя не изменялось

В случае если при создании пользователь пытается создать папку с именем, которое уже существует, появляется сообщение «В данном месте уже существует папка с таким именем. Заменить?». Пользователь выбирает «Да» или «Нет». (Рисунок 27)

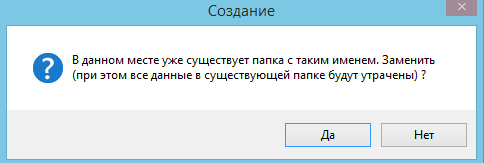


Рисунок 27 – создание существующей папки

Если пользователь в случае с поиском не выбирает место где искать и не выбирает хотя бы один из критериев поиска, то в результате поиска выводится пустое окно. (Рисунок 28)

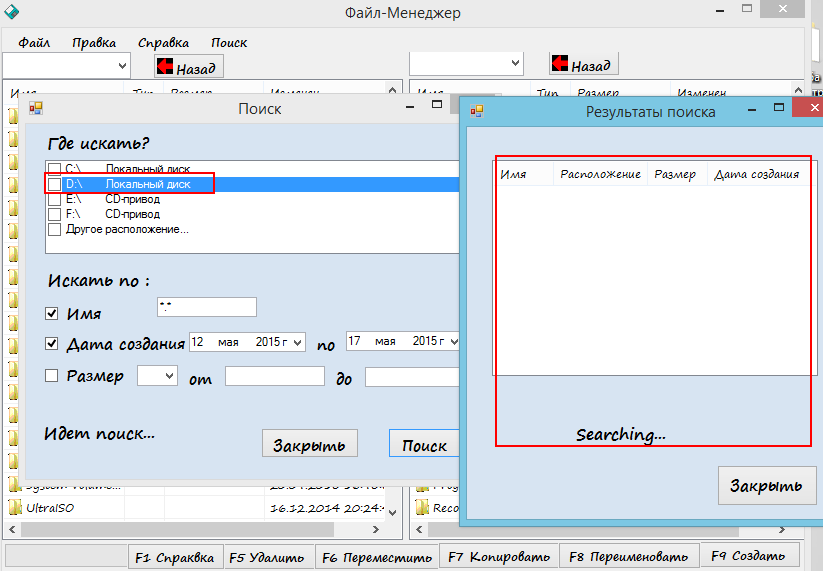


Рисунок 28 – не выбрано где искать

При тестировании функции поиска, было выявлено, что пользователю приходить ждать некоторое время чтобы получить желаемый результат. По сравнению с выполнением поиска в программе Total Commander пользователь получает результаты непосредственно во время поиска. (Рисунок 29)

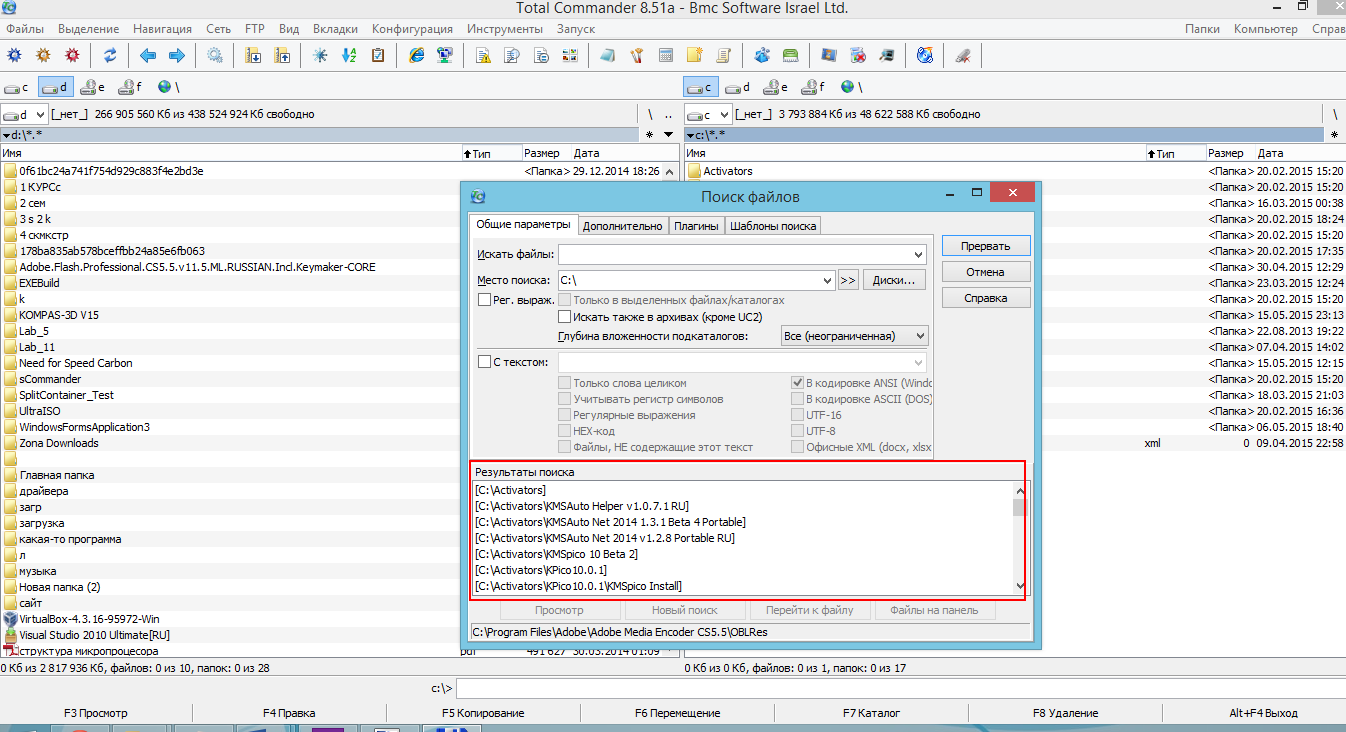


Рисунок 29 – поиск в Total Commander

По результатом тестирования можно сделать вывод, что приложение имеет достаточно простой интерфейс, пользователь легко может найти необходимые ему функции. При совершении пользователем различных ошибок программа выдает соответствующие сообщения, благодаря которым пользователь сразу понимает, что он сделал не так. Пользователю так же помогают всплывающие подсказки при наведении на кнопки. Также в ходе тестирования был выявлен недостаток. Во время поиска пользователь приходиться ждать некоторое время пока программа выведет результаты.

# Заключение

Язык C# очень актуален в настоящее время, предоставляет намного больше возможностей, нежели Delphi к примеру. Писать в C# намного удобнее и проще. Знание си подобного синтаксиса значительно упрощает изучение других языков, таких как Java, c++, c, pithon, а также изучение скриптов php и прочего.

В курсовой работе описана и реализована программа, написанная с помощью C#, похожая на Total Commander.

В результате выполнения этой работы были получены практические навыки работы с файлами и каталогами. Данная программа предоставляет возможность копировать, удалять, перемещать, переименовать файлы и папки. Также разработан поиск файлов на некотором системном устройстве. Интерфейс приложения получился простым и интуитивно-понятным пользователю.

Для дальнейшей доработки этой программы необходимо сначала в течение некоторого времени применять её для оценки эффективности, а потом добавлять или исправлять все недочеты, которые пропустил или не посчитал нужным реализовывать разработчик.

# Список используемой литературы и источников

1. Шилдт, Г. Полный справочник по C# /Г. Шилдт – ООО “И.Д. Вильямс”,2004 – 752 с.

2. Мак-Дональд Microsoft ASP.NET 4 с примерами на C# 2010/ Мэтью Мак-Дональд – ООО “И.Д. Вильямс”,2011.-1024с.

3. Дэвис С.Р., Сфер Ч. C# 2005 для "чайников".

4. <https://msdn.microsoft.com>

5. <http://professorweb.ru/>

6. <http://habrahabr.ru/>

7. Фролов А. В., Фролов Г. В. Визуальное проектирование приложений C# - КУДИЦ-Образ, 2003 – 353 с.