

## СОДЕРЖАНИЕ

Введение .....	5
1 Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству .....	7
1.1 Анализ существующих прототипов .....	7
1.2 Постановка задачи .....	10
2 Анализ требований к программному средству и разработка функциональных требований.....	11
2.1 Описание функциональности программного средства .....	11
2.2 Спецификация функциональных требований .....	12
3 Проектирование программного средства.....	13
3.1 Алгоритм прорисовки файлового меню .....	13
3.2 Класс работы с файловой системой. Обработка ошибок.....	16
4 Создание программного средства.....	19
5 Тестирование, проверка работоспособности и анализ полученных результатов ..	21
5.1 Тестирование, проверка работоспособности и анализ полученных результатов .....	21
5.2 Ввод строки.....	25
5.3 Удаление папки .....	26
5.4 Предпросмотр файла.....	26
5.5 Просмотр дисков .....	27
5.6 Вывод из прохождения тестирования .....	28
6 Руководство по установке и использованию .....	29
6.1 Предустановки.....	29
6.2 Запуск файлового менеджера .....	32
6.3 Использование файлового менеджера .....	34
6.4 Частые ошибки .....	37
Заключение.....	39
Список использованных источников.....	40
Приложение А.....	41
Приложение Б .....	58

## ВВЕДЕНИЕ

Файловый менеджер – компьютерная программа, предоставляющая пользовательский интерфейс для работы с файловой системой и файлами. Файловый менеджер позволяет выполнять наиболее частые операции с файлами – создание, просмотр, перемещение, переименование, копирование и удаление.

Файловые менеджеры начинают свою историю со специальных программ – оболочек, разработанных для операционной среды DOS. В свое время эти программы были настоящим прорывом в IT-индустрии. Они позволили значительно расширить возможности ПК и способствовали массовой интеграции компьютеров во все отрасли делопроизводства. Первым, ставшим самым известным файловым менеджером, стал «Norton Commander». Гениальный двухпанельный интерфейс и синий цвет для многих стал синонимом «работающего компьютера».

Другим типом файловых менеджеров также выделяют навигационный, который имел только одно окно и чаще всего использовался в консольном приложении. В таком виде пользователь в любой момент времени видел только одну директорию, а взаимодействие с файлами и папками происходило через больше различных сочетаний клавиш, по сравнению с двухпанельными.

На данный момент времени оба типа файловых менеджеров, описанных ранее, считаются устаревшими, большая часть пользователей использует пространственный тип, а некоторые даже и не видели другие. Однако некоторые люди, заставшие двухпанельные файловые менеджеры, в пик их популярности, отмечают, что они являются наиболее удобными.

Тем не менее, файловые менеджеры других типов все еще считаются востребованными, в некоторых отраслях IT индустрии, так как доступ к графическому интерфейсу не всегда является возможным. Например, при подключении к удаленному серверу, которое происходит чаще всего по SSH протоколу, доступ к среде рабочего стола хоть и является возможным, однако может потребовать значительное время для его настройки. К тому же, в преобладающем большинстве случаев это не является необходимым или желательным, так как:

- а) подключение происходит через специальное программное обеспечение, которое уже включает в себя среду с файловым менеджером, необходимым для полноценную работу с ресурсом;
- б) продвинутые пользователи обходятся без файлового менеджера, с помощью встроенных команд.

Хоть и консольный файловые менеджеры не среди являются популярными среди обычных пользователей, некоторые продвинутые

отмечают, что такие программы ускоряют работу с файловой системой и избавляют от рутинной работы. Таким образом можно выделить основные характеристики хорошего консольного файлового менеджера:

- а) малый вес – при работе на удаленных ресурсах, даже в нынешнее время, ограничение по памяти является весомой причиной выбора того или иного программного обеспечения;
- б) быстрота – с точки зрения пользования, если ПО не ускоряет, а лишь замедляет ту работу, что можно было выполнить с помощью двух-трех команд, то оно не является оптимальным;
- с) удобство – заменяя работу с написания команд на нажатие сочетаний клавиш, необходимо, чтобы они были интуитивно понятными и удобными для исполнения.

Данная работа является консольным файловым менеджером нацеленной на операционную систему Windows. Для работы с файловой системой был выбран язык C++17 со встроенной библиотекой `filesystem`, что значительно ускоряет работу и уменьшает количество необходимых для работы файлов до одного; библиотека `WinApi` для вывода информации на экран, что также ускоряет работу программы.

Главными задачами этой работы является создание оптимизированного файлового менеджера, который, помимо того, что должен быть быстрым, еще должен быть удобен и интуитивно понятен. В угоду этим правилам будет совершено некоторое ограничение возможностей ПО – оно будет урезано до необходимого минимума, т.е.:

- а) создание файлов/папок;
- б) удаление файлов/папок;
- с) переименование;
- д) перемещение по директориям;
- е) копирование и перемещений файлов/папок;
- ф) предпросмотр файлов.

Целью данного курсового проекта является разработка программного средства «madfm».

# 1 АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ

## 1.1 Анализ существующих прототипов

Найти аналоги к данной работе является сложной задачей, так как консольные файловые менеджеры не являются популярными для операционной системы Windows, а существующие либо совсем не удовлетворяют требованиям программного средства, либо являются платными. Наиболее подходящие приложения удалось отыскать на популярном ресурсе открытых репозиторий – Github.

### 1.1.1 File Manager by Anton Tymoshchuk

Консольное приложение написанное на Python, с использованием библиотеки rounput, для обработки нажатий клавиш.



Рисунок 1.1 – File Manager by Anton Tymoshchuk

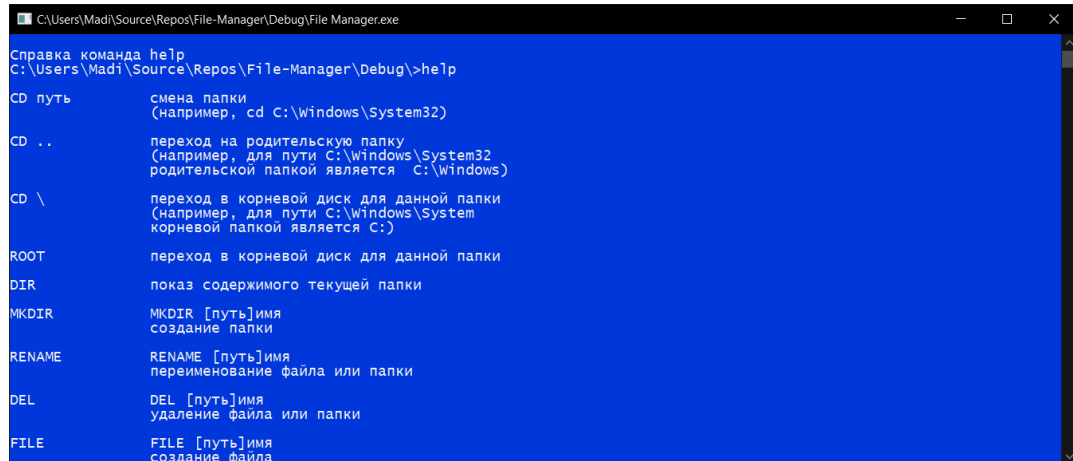
Рисунок 1.1 полностью иллюстрирует весь интерфейс приложения – выбранная директория обозначается знаком больше, файлы обозначаются заглавной латинской буквой F. Управление осуществляется стрелками и кнопками Enter и Escape, а также некоторыми сочетаниями клавиш.

Из плюсов данной программы можно отметить лишь ее кроссплатформенность, а к минусам относятся:

- a) медленная скорость работы;
- b) переполнение консольного окна;
- c) не интуитивное управление;
- d) отсутствие возможности предпросмотра файлов, их копирование и перемещение.

### 1.1.2 File-Manager by wwwIgorNet

Приложение написанное на языке C++, с поддержкой библиотеки Windows. Используя библиотеку WinApi и обширные возможности языка, удалось достичь высокой скорости отклика приложения, а также обработку различных ошибок, однако вся информация выводится через стандартный поток вывода, что значительно замедляет всю работу программного средства, оставляя его бездейственным, во время загрузки директорий, с большим количеством субдиректорий.



```
C:\Users\Madi\Source\Repos\File-Manager\Debug\File Manager.exe
Справка команда help
C:\Users\Madi\Source\Repos\File-Manager\Debug>help

CD путь      смена папки
               (например, cd C:\windows\System32)

CD ..         переход на родительскую папку
               (например, для пути C:\windows\System32
               родительской папкой является C:\windows)

CD \          переход в корневой диск для данной папки
               (например, для пути C:\windows\System
               корневой папкой является C:)

ROOT          переход в корневой диск для данной папки

DIR           показ содержимого текущей папки

MKDIR         MKDIR [путь]имя
               создание папки

RENAME        RENAME [путь]имя
               переименование файла или папки

DEL           DEL [путь]имя
               удаление файла или папки

FILE          FILE [путь]имя
               создание файла
```

Рисунок 1.2 – File-Manager by wwwIgorNet

Как видно из рисунка 1.2 все взаимодействие выполнено через командную строку, с расширенными возможностями, по сути дела эмулируя команды unix систем.

Плюсы данного ПО:

- а) быстрота работы;
- б) полный набор базовых функций

К минусам же относятся:

- а) медленный вывод информации на экран;
- б) переполнение окна;
- с) отсутствие возможности предпросмотра;
- д) не стандартизированный вывод.

### 1.1.3 FileManager by Vanya Score

Приложение написанное на С# уже является быстрым для операционной системы Windows, однако это ПО также использует другие техники для ускорения его работы.

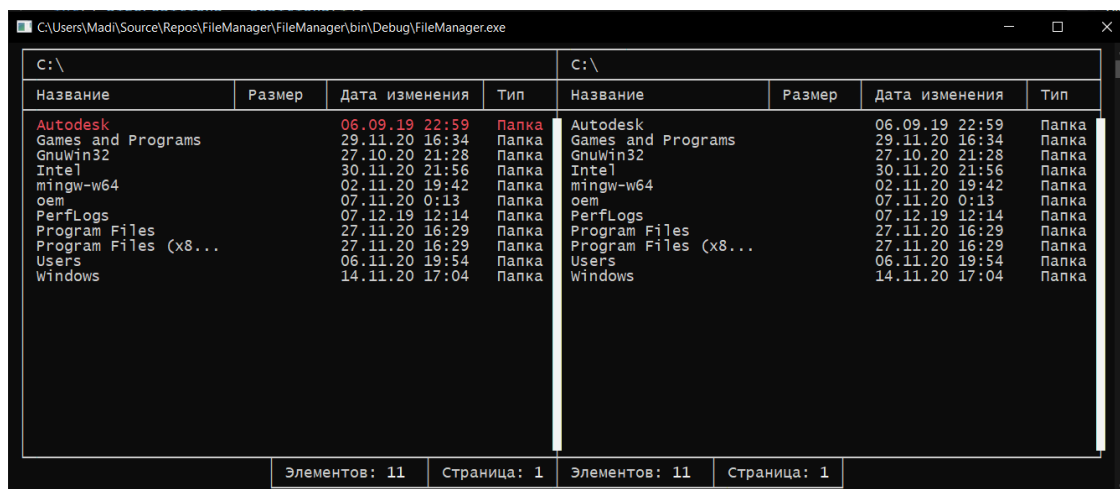


Рисунок 1.3 – FileManager by Vanya Score

Из рисунка 1.3 видно, что этот файловый менеджер относится к двухпанельному типу. С помощью кодировки Unicode и библиотеки WinApi, было достигнута максимально понятная и интуитивная среда, в которой может разобраться даже неопытный пользователь.

Плюсы FileManager:

- а) превосходная скорость работы;
- б) понятный и удобный интерфейс;
- с) малое потребление ресурсов;
- д) русский язык интерфейса.

Минусы:

- а) отсутствует возможность какого либо взаимодействия с папками и файлами (помимо их удаления);
- б) неправильная начальная директория (указывала на диск E).

## 1.2 Постановка задачи

Основной задачей этой работы является создание консольного файлового менеджера, который должен иметь базовый необходимый набор функций, а также передвижение по директориям диска и самим дискам.

Данная программа должна обеспечивать следующие функции:

- a) графическая визуализация пунктов меню (содержимого директории), включающее в себя:
  - 1) дата и время последнего редактирования;
  - 2) размер (для файлов);
  - 3) название;
  - 4) дифференцирование файлов и директорий (различный цвет для каждой категории);
- b) перемещение по директориям;
- c) создание директорий и файлов;
- d) удаление директорий и файлов;
- e) копирование директорий и файлов;
- f) переименование директорий и файлов;
- g) предпросмотр файлов:
  - 1) текстовый (одна страница текста);
  - 2) бинарный (hexdump);
- h) отображение меню помощи с управлением;
- i) индикация ошибок.

## **2 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ**

### **2.1 Описание функциональности программного средства**

Данная программа запускается в консольном режиме. Язык интерфейса – английский. Начальная отображаемая директория является директорией запуска. В самом верху отображается рабочая директория. Ниже расположена таблица состоящая из трех не подписанных полей – последнее время редактирования, размер и название. Управление осуществляется с помощью стрелок, буквенной клавиатуры, а также функциональной клавишей F2. При выполнении любых действий появляется соответствующая надпись. При выполнении запрещенных и/или ошибочных действий появляется информация об ошибке.

#### **2.1.1 Переименование и создание**

Переименование и создание файлов и папок сопровождается надписью с вводом нового имени. Данное действие не требует подтверждения, однако, если папка или файл с таким названием уже существует в рабочей директории, то выводится соответствующая информация об ошибке, а действие отменяется.

#### **2.1.2 Удаление**

Удаление является действием с обязательным подтверждением. Для предотвращения случайного удаления, подтвердить это действие возможно лишь специальной клавишей, нажатие любой другой отменяет действие.

#### **2.1.3 Копирование и перемещение**

При копировании и/или перемещении файлов или папок, первым делом происходит копирование пути из рабочей директории во внутренний буфер, следовательно в другой директории при нажатии специальной клавиши происходит копирование/перемещение сохраненного элемента файловой системы, с последующей очисткой буфера. Данное действие не требует подтверждения.



## 2.2 Спецификация функциональных требований

Во время разработки данного программного средства должны быть реализованы следующие функции:

- a) управление элементами файловой системы;
- b) отображение информации в консоли (без полной ее перерисовки), включающее в себя:
  - 1) отображение файлов и папок;
  - 2) отображение ошибок;
  - 3) отображение сообщений;
  - 4) пользовательский ввод;
- c) обработка ошибок;
- d) обработка нажатий.

## **3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

### **3.1 Алгоритм прорисовки файлового меню**

При создании графического меню в консоли, необходимо было, чтобы в рабочей директории выбранный элемент подсвечивался, а остальные оставались на черном фоне (или на том фоне, который был при запуске приложения). Также, помимо того, что количество элементов должно быть ограниченным, это как то должно отображаться. Все это приводит к тому, что из-за, примерно, 12-15 консольных строк с полезной информацией, приходится обновлять все консольное окно. Это несомненно вело не только к неприятному для глаз мерцанию, а также значительно замедляло процесс работы программного средства, которое в свою очередь большую часть только и делает, что отображает текст на экране.

В качестве решения данной проблемы были выбраны два пути, которые по одиночке уже ускоряли прорисовку в 2-3 раза, а вместе полностью исключили фактор неприятного торможения и мерцания – буферизация текста в консольном окне (в отдельном массиве строк) и перерисовка лишь тех строк, что были изменены.

Для этого был создан класс-структура ConsoleLine, который хранит два поля – строковую репрезентацию консольной строки и ее цвет (атрибут WinApi). В модуле прорисовки меню есть два массива – хранящие предыдущие и последующие строки. Он также хранит индекс выбранного элемента и индекс начала страницы (если количество элементов больше 15).

При любом изменении информации (изменение индекса, перемещение по директориям и т.д.) происходит перерисовка меню в следующих этапах:

- а) очистка массива следующих строк;
- б) добавление элементов меню:
  - 1) строка с рабочей директорией;
  - 2) опциональная строка с номером страницы;
  - 3) строки с файлами и информации о них;
  - 4) опциональная строка с номером страницы (второй раз);
- с) сравнение новых строк с существующими;
- д) если строки не равны – их перерисовка
- е) изменение массива нынешних строк.

Таким образом даже при большом и частом изменении индекса, быстрым переходом между директорий, изменением страницы экран не мерцает, а прорисовка новых элементов происходит незаметно для глаза. Ниже предоставлен более детальный алгоритм, с привязкой к WinApi.

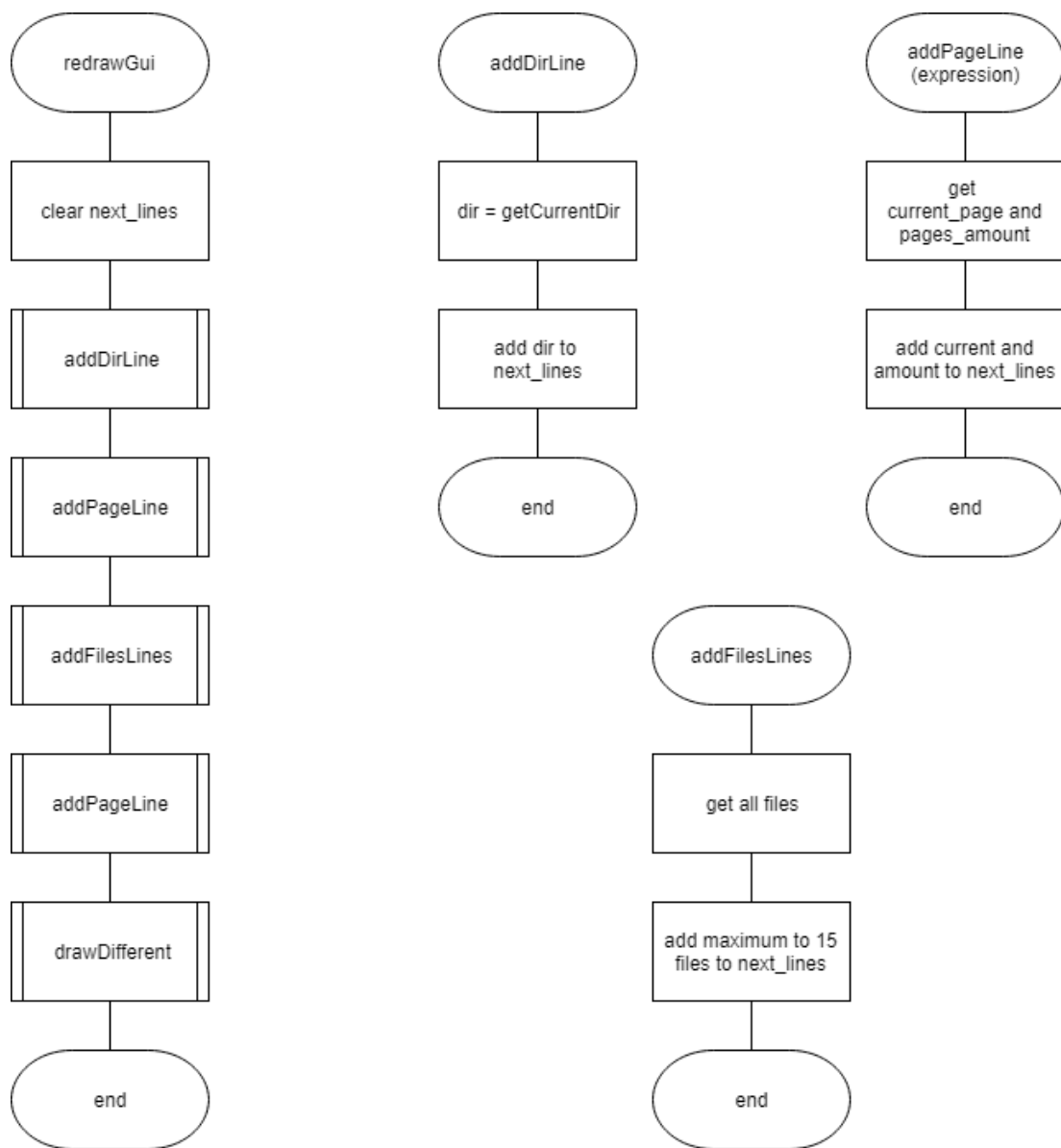


Рисунок 3.1 – Блок-схема основной части прорисовки меню

```

1. // перерисовка
2. void MenuDrawer::drawDifferent() {
3.     // проход по всем строкам
4.     for (int i = 0; i < next_lines.size(); i++) {
5.         // если новая строка не равна старой
6.         if (current_lines[i] != next_lines[i]) {
7.             // сохраняем ее координату
8.             COORD current_line_coord = { 0, i };
9.             // устанавливаем курсор на эту строку
10.            SetConsoleCursorPosition(cgh->h_console, current_line_coord);
11.            // очищаем строку
12.            cgh->utils.clearCurrentLine(); // имплементация ниже
13.            // выводим строку на экран
14.            cgh->utils.outputLine(next_lines[i]); // имплементация ниже
15.            // сохраняем новую строку как старую
16.            current_lines[i] = next_lines[i];
17.        }
18.    }
19. }

```

```

20. void ConsoleGuiUtils::clearCurrentLine() {
21.     // получаем информацию о буфере консоли
22.     CONSOLE_SCREEN_BUFFER_INFO s;
23.     GetConsoleScreenBufferInfo(h_console, &s);
24.     // сохраняем текущую позицию курсора и ширину консоли
25.     COORD current_line_coord = { 0, s.dwCursorPosition.Y };
26.     DWORD w, cells = s.dwSize.X;
27.     // очищаем строку и цвет
28.     FillConsoleOutputCharacter(h_console, ' ', cells, current_line_coord, &w);
29.     FillConsoleOutputAttribute(h_console, 0, cells, current_line_coord, &w);
30. }
31. void ConsoleGuiUtils::outputLine(ConsoleLine line) {
32.     CONSOLE_SCREEN_BUFFER_INFO s;
33.     GetConsoleScreenBufferInfo(h_console, &s);
34.     COORD current_cursor_coord = s.dwCursorPosition;
35.     COORD next_line_coord = { 0, (short)(s.dwCursorPosition.Y + 1) };
36.     DWORD w;
37.     // выводим строку и цвет
38.     WriteConsoleA(h_console, line.c_str(), line.getText().size(), &w, nullptr);
39.     FillConsoleOutputAttribute(h_console, line.getAttribute(), line.size(),
                                current_cursor_coord, &w);
40.     // переносим курсор на следующую строку
41.     SetConsoleCursorPosition(h_console, next_line_coord);
42. }

```

## 3.2 Класс работы с файловой системой. Обработка ошибок

Для работы с файлами и директориями операционной системы был выбран класс `filesystem`, который включен в стандартную библиотеку C++, начиная с 17 версии, который работает по стандартам POSIX. Все функции данной библиотеки имеют перегрузку с возвращаемым кодом ошибки, без вызова исключения. Это, помимо того, что делает код более читабельным, освобождает от необходимости обрабатывать исключения. Данный подход был выбран, так как, согласно общепринятому соглашению, исключения должны происходить лишь в тех случаях, когда код работает не правильно, тем самым обращая внимание на возможный баг. Однако, предоставляя возможность пользователю менять исходные данные, ошибка не будет считаться багом, а обработка ввода должна происходить незамедлительно. Таким образом не нарушается целостность программы, а ее скорость может заметно возрасти.

Чтобы каждый раз не обрабатывать код ошибки, в вызывающей функции, был создан нумерованный класс `FiledirectoryException`, каждый тип в котором, соответствует определенной ошибке, что позволяет легче их обрабатывать и выводить соответствующие сообщения. Следовательно, каждая функция имеет тип возвращаемого значения `FiledirectoryException`, который, в том числе имеет тип `NoException`, что говорит о безошибочном выполнении подпрограммы.

Единственным ограничением этой библиотеки является отсутствие возможности создавать файл, поэтому для этих целей был выбран файловый поток `fstream`, из стандартной библиотеки C++. Он по умолчанию не вырабатывает никакие исключения, а лишь записывает код ошибки в отдельную глобальную переменную. Ниже более подробно описаны функции этого класса, с привязкой к стандартной библиотеке.

### 3.2.1 Переименование и перемещение файла

Оба этих действия, согласно стандарту, выполняется одной функцией – `move`. Главное отличие состоит в том, что при перемещении файла или папки различаются пути, но названия сохраняются (в общем случае), а при переименовании – сохраняются пути, меняются названия. Семантически эквивалентной командой в Unix системах являются:

```
mv old_path new_path // перемещение
mv path/old_name path/new_name // переименование
```

Ошибку в данной команде могут вызвать – зарезервированные имена Windows и ошибка доступа. Оба этих исключения запишут уникальный код в переменную. Проверка на существующий файл должна происходить вызывающей функцией.

```
1. FileDirectoryException FileDirectory::move(const std::string &old_path, const std
   ::string &new_path) {
2.     // код ошибки будет храниться тут
3.     std::error_code ec;
4.     // rename == move (POSIX)
5.     fs::rename(old_path, new_path, ec);
6.     // если произошла ошибка значение ec будет не равно нулю
7.     if(ec.value() != 0)
8.         return FileDirectoryUtils::handleExceptionCode(ec.value());
9.     return FileDirectoryException::NO_EXCEPTION;
10.}
11.
12. FileDirectoryException FileDirectory::changeName(const std::string& old_name,
   const std::string& new_name) {
13.     // старое и новое имя - одинаковый путь
14.     auto old_p = fs::current_path() / old_name;
15.     auto new_p = fs::current_path() / new_name;
16.     // вызов move
17.     return move(old_p.string(), new_p.string());
18.}
```

### 3.2.2 Создание директории. Удаление и копирование

Создание директории происходит по стандарту POSIX командой `mkdir`, которая относится как функция `create_directory` в библиотеке `filesystem`. Ошибки обрабатываются также, как и в перемещении.

Удаление выполняется функцией `remove`, что соотносится к стандартной функции `remove`. Копирование выполняется функцией `copy`, которая выполняет глубокое копирование для операционной системы Windows (т.е. создает копию в полном ее понимании, а не создает копию ссылки), при этом не меняет названия файла. Функция может вызвать исключение лишь в случае отсутствия прав. Данные команды семантически эквивалентны таким командам Unix:

```
mkdir path                // создает папку
rm path                   // удаляет
cp old_path new_path      // копирует
```

### 3.2.3 Создание файла

Как упоминалось ранее, в данной библиотеке не предусмотрена возможность создавать файл. Для этих целей был выбран выходной файловый поток `ofstream`, который создает/открывает файл и сразу же его закрывает. Затем проверяется код ошибки и если он не равен нулю, то файл был успешно

создан. Так как по умолчанию эти потоки исключения не вызывают, то в использовании других функций пропадает необходимость. Ошибку в данном случае может вызвать зарезервированное имя или отсутствие прав. Проверку на существование файла должна делать вызывающая функция.

```
1. FileDirectoryException FileDirectory::createFile(const std::string &name) {
2.     // получение пути файла
3.     auto path = fs::current_path() / name;
4.     // создание файла
5.     std::ofstream file;
6.     file.open(path.string());
7.     file.close();
8.     // проверка на существование ошибки
9.     if(errno != 0)
10.         return FileDirectoryUtils::handleExceptionCode(errno);
11.     return FileDirectoryException::NO_EXCEPTION;
12. }
```

## 4 СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА

В ходе разработки программного средства исходный код был поделен на 14 классов:

- a) File – хранит информацию об отдельном элементе файлового пространства (в том числе и о директориях);
- b) Filedirectory – функции для работы с файлами;
- c) FiledirectoryUtils функции для обработки информации о файлах;
- d) FileCreator – модуль меню – создает файлы;
- e) FileDeleter – модуль меню – удаляет файлы;
- f) FileMover – модуль меню – перемещает файлы;
- g) FilePreviewer – модуль меню – отображает предпросмотр файла;
- h) FileRenamer – модуль меню – переименовывает файлы;
- i) HelpDrawer – модуль меню – отображает информацию об управлении;
- j) MenuDrawer – модуль меню – отображает меню;
- k) ConsoleGuiHandler – само меню;
- l) ConsoleGuiUtils – обертка WinApi для вывода информации на консоль;
- m) ConsoleLine – хранит информацию о каждой отображаемой линии в меню;
- n) KeypressHandler – обрабатывает нажатия клавиш.

Внутри каждого класса присутствуют вспомогательные приватные методы, для решения определенных задач. Основной класс с обработкой меню был поделен на модули – вспомогательные классы, каждый решающий определенную задачу. Таким образом было достигнуто логическое разбиение программы, что облегчает ее разработку и уменьшает количество случайных ошибок. Ниже приведены основные методы классов:

- a) Filedirectory:
  - 1) move, copy, changeName, createDir, createFile, deleteFile были детально описаны в секции 3.2;
  - 2) containsCurrent – возвращает true, если рабочая директория содержит папку или файл с определенным названием;
  - 3) reinit – переопределяет рабочую директорию;
- b) FiledirectoryUtils:
  - 1) parseName, parseSize, parseTime – преобразует название, размер и время последнего редактирования под один стандарт, для последующего отображения на экран;



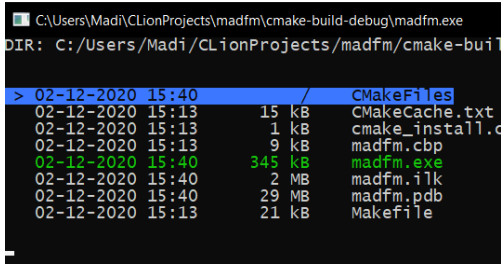
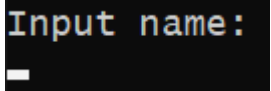
- 2) `handleExceptionCode` – возвращает один из нумерованных типов, отображающий определенную ошибку (например `FileDirectoryException::FILE_NOT_FOUND`);
- c) `FileCreator` (с обработкой ошибок):
  - 1) `createFile` – с помощью класса `FileDirectory` создает файл и отображает его в меню;
  - 2) `createDir` – создает и отображает новую папку;
- d) `FileDeleter` (с обработкой ошибок):
  - 1) `deleteFile` – удаляет файл и обновляет меню;
- e) `FileMover` (с обработкой ошибок):
  - 1) `savePath` – сохраняет путь выбранного файла во внутренний буфер;
  - 2) `moveFile` – перемещает файл из буфера в рабочую директорию;
  - 3) `copyFile` – копирует файл из буфера в рабочую директорию;
- f) `FilePreviewer` (с обработкой ошибок):
  - 1) `showTextPreview` считывает выбранный файл и отображает одну страницу на экран;
  - 2) `showRawPreview` – считывает выбранный файл и форматированно отображает его, побайтово (hexdump);
- g) `FileRenamer` (с обработкой ошибок):
  - 1) `renameFile` – переименовывает файл и обновляет меню;
- h) `HelpDrawer`:
  - 1) `showHelp` отображает информацию помощи на экран;
- i) `MenuDrawer`:
  - 1) `redrawGui`, `addDirLine`, `addPageLine`, `addFilesLines`, `drawDifferent` были описаны детально в секции 3.1;
  - 2) `cleanRedrawGui` – перерисовывает меню, при этом очищая буфер со строками;
- j) `ConsoleGuiHandler` – является оберткой для модулей, все методы этого класса вызывают соответствующие методы в модулях;
- k) `ConsoleGuiUtils`:
  - 1) `clearScreen` очищает консоль;
  - 2) `clearLine` – очищает одну строку в консоли;
  - 3) `outputLine` – выводит одну строку в консоли;
  - 4) `outputChar` – выводит один символ в консоли;
  - 5) `inputLine` считывает текст, введенный с клавиатуры
- l) `KeyPressHandler`:
  - 1) `start` начинает слушать нажатия клавиш и вызывает соответствующие методы в `ConsoleGuiHandler`.

## 5 ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

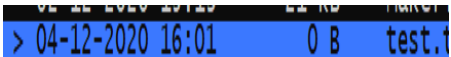
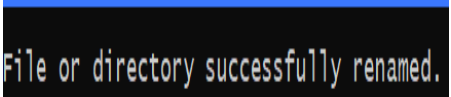
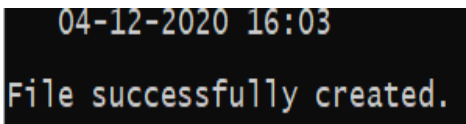
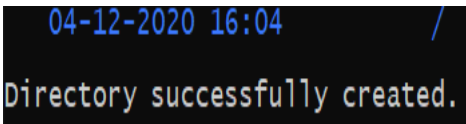
### 5.1 Тестирование, проверка работоспособности и анализ полученных результатов

Тестирование программного средства проходило на персональном компьютере с установленной операционной системе Windows 10.

Таблица 5.1 – Тестирование программы

№	Тестируемая функциональность	Последовательность действий	Ожидаемый результат	Полученный результат
1	Запуск	Двойной щелчок по исполняемому файлу	Консольное окно очищается; Выводится список данной директории	 <p>Тесты пройдены.</p>
2	Запуск (через команду)	Запуск по названию ПС (madfm)		
3	Отображение диалогового сообщения ввода	Нажатие r/R; Нажатие n/N;	Вывод сообщения «Input name:»	 <p>Сообщение вывелось корректно. Тест пройден.</p>
4	Скрытие диалогового сообщения ввода	Нажатие r/n/R/N; Нажатие Esc	Скрытие сообщения «Input name»; Скрытие введенной строки	Сообщение скрылось корректно. Введенная строка скрылась корректно. Тест пройден.
5	Ввод строки	Нажатие r/n/R/N; Ввод строки	Вывод; Нажатие backspace удаляет символ	Сообщение вводится корректно. Нажатие backspace переносит каретку назад, но символ не удаляет. <b>Тест не пройден.</b>

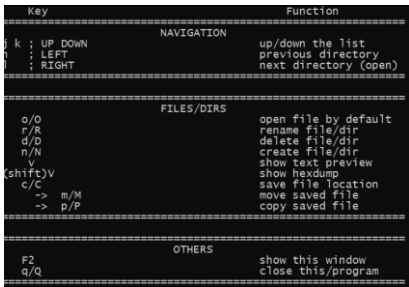
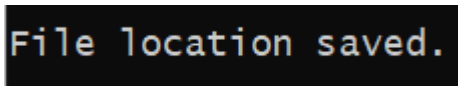
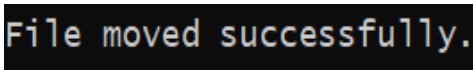
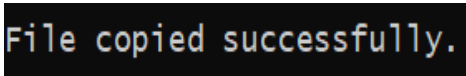
Продолжение таблицы 5.1

№	Тестируемая функциональность	Последовательность действий	Ожидаемый результат	Полученный результат
6	Ограничение ввода	Нажатие г/n/R/N; Ввод строки	Строка ограничена 64 символами	После ввода 64 символов ПС перестает реагировать на нажатия клавиш. Тест пройден.
7	Переименование файла	Нажатие г/R; Ввод названия;	Выделенный элемент успешно переименовывается; Вывод сообщения об успешном переименовании	  Выделенный элемент успешно переименовался. Тест пройден.
8	Переименование папки	Нажатие Enter		
9	Создание файла	Нажатие n/N; Ввод названия; Нажатие f/F	Создание файла; Вывод сообщения об успехе	 В рабочей директории создается файл с введенным названием. Тест пройден.
10	Создание папки	Нажатие n/N; Ввод названия; Нажатие d/D	Создание папки; Вывод сообщения об успехе	 В рабочей директории создается папка с введенным названием. Тест пройден.
11	Отображение диалогового сообщения подтверждения	Нажатие d/D	Вывод сообщения «Are you sure you want to delete this? (y/N)»	Сообщение выводится успешно. Тест пройден

Продолжение таблицы 5.1

№	Тестиру- емая фун- кцио- нальность	Последова- тельность действий	Ожидаемый результат	Полученный результат
12	Скрытие сообщения подтвер- ждения	Нажатие d/D; Нажатие любой клавиши кроме у	Сообщение скрывается	Сообщение успешно скрылось. Удаление не произведено. Тест пройден.
13	Удаление файла	Нажатие d/D; Нажатие у	Файл удаляется	Файл удаляется успешно. Тест пройден.
14	Удаление папки	Нажатие d/D; Нажатие у	Папка удаляется	Папка не удаляется, выводится сообщение об ошибке. <b>Тест не пройден.</b>
15	Открытие файла в приложе- нии по умолча- нию	Нажатие o/O	Файл открывается в приложении по умолчанию	Текстовый файл открывается в блокноте. Файлы с неизвестным расширением открываются в редакторе. Фотографии открываются в средстве просмотра изображений и т.д. Тест пройден.
16	Открытие папки в приложе- нии по умолча- нию	Нажатие o/O	Меняется рабочая директория	При попытке открыть папку в приложении по умолчанию в ПС изменятся рабочая директория на эту папку. Тест пройден.
17	Просмотр файлов в текстовом виде	Нажатие v	Открывается средство предпро- смотра текстового содержимого	Одна страница текста выводится на экран успешно, однако это занимает значительное время. <b>Тест не пройден.</b>
18	Hexdump	Нажатие shift+V	Открывается средство просмотра байтов файла	Одна страница выводится успешно, однако не совпадает размер файла и занимает значительное время. <b>Тест не пройден.</b>

Продолжение таблицы 5.1

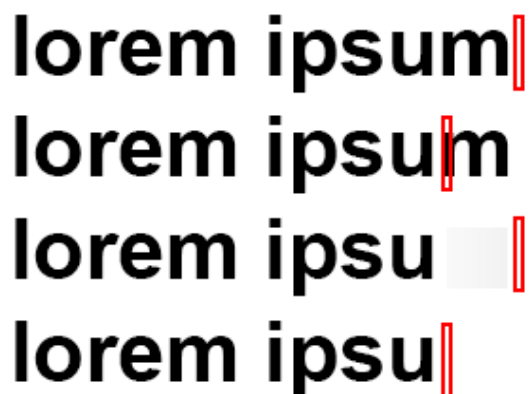
№	Тестируемая функциональность	Последовательность действий	Ожидаемый результат	Полученный результат
19	Отображение помощи	Нажатие F2	Вывод помощи на экран	 <p>Тест пройден.</p>
20	Отображение сообщения сохранения пути	Нажатие с/С	Вывод сообщения «File location saved»	 <p>Тест пройден</p>
21	Перемещение файла	Нажатие с/С; Изменение рабочей директории; Нажатие м/М	Успешное перемещение в другую директорию; Вывод сообщения об успехе	 <p>Файл и папка успешно перемещена в другую директорию, с сохранением всего содержимого. Тест пройден.</p>
22	Перемещение папки			
23	Копирование файла	Нажатие с/С; Изменение рабочей директории; Нажатие р/Р	Успешное копирование в другую директорию; Вывод сообщения об успехе	 <p>Файл и папка успешно скопирована в другую директорию, с сохранением всего содержимого. Тест пройден.</p>
24	Копирование папки			
25	Перемещение по директориям	Нажатие стрелок и/или hjkl	Перемещение во все возможные директории	Программа успешно меняет рабочую директорию и индексирует все ее содержимое, однако нет возможности сменить диск. <b>Тест не пройден.</b>

Все данные тесты были проведены с учетом обработки ошибок, следовательно, если указано, что тест пройден, то и ошибки выводятся корректно.

При тестировании было выявлено, что в тестах 21-24 выводится корректное, однако не точное сообщение, которое было заменено на более осмысленное. Остальные не пройденные тесты требуют отдельного рассмотрения.

## 5.2 Ввод строки

Ввод строки происходит с помощью использования библиотеки расширения компилятора `conio.h`. Конкретно используется функция `getch()`, которая считывает символы с буфера клавиатуры, однако не выводит их на экран. Данная функциональность была использована для обработки специальных клавиш `Esc` и `Enter`, которые выполняют выход и сохранение ввода, соответственно. Однако при обработке нажатия, клавиша `Backspace` не относилась к отдельно рассматриваемым, следовательно ее символ выводился напрямую в консоль, что лишь изменяло позицию каретки, но не удаляло символ. Новая функциональность была достигнута, с помощью двойного переноса каретки. Единичное нажатие `Backspace` вызывает последовательность действий, изображенных и описанных ниже.



lorem ipsum

lorem ipsum

lorem ipsu

lorem ipsu

Рисунок 5.1 – Последовательность удаления символа

- а) Перемещение каретки назад, при помощи вывода символа;
- б) Вывод пустого (пробельного) символ;
- с) Перемещение каретки назад.

Вместе с этим из буфера удаляется последний символ (заменяется на нулевой).

### 5.3 Удаление папки

При попытке удалить пустую папку не возникает никаких ошибок, однако при удалении папки, содержащую другие файлы, возникает не обработанная ошибка и сообщение «An error has occurred». Данные вещи указывают на то, что библиотека не может удалить рекурсивно при помощи функции `remove`. Проблема решилось заменой этой функции на функцию `remove_all`, которая одинаково с предыдущей удаляет файлы, но при этом папки удаляет рекурсивно. Данная команда семантически эквивалентна команде Unix:

```
rm -r path // удаляет (-r - рекурсивно)
```

### 5.4 Предпросмотр файла

Данная функциональность содержит две ошибки – вывод содержимого занимает заметно большое время, даже при малом размере файла и некорректный вывод файла при `hexdump`. Рассмотрим их по отдельности.

#### 5.4.1 Буферизация вывода

Как уже упоминалось ранее, самой затратной для программного средства операцией является вывод информации. Так как для вывода одной страницы содержимого файла нет необходимости считывать весь файл (в обратном случае это замедляло бы работу, при этом даже никак не использовалось), то чтобы буферизировать содержимое необходимо знать размеры страницы. Для этого запросом `GetConsoleScreenBufferInfo` достается информация о количестве строк и столбцов в консольном окне. Исходя из этой информации определяется количество символов, которые необходимо считать (причем надо учитывать, что количество строк строго **не равно** количеству символов новой строки) и полностью заполняется буфер. Только после закрытия файла буфер выводится на экран. Имплементацию этого кода можно посмотреть в приложении А: файл – `FilePreviewer.cpp`, метод – `showTextPreview`. Блок-схему для вывода байтов можно посмотреть в приложении Б. Таким образом достигается большая скорость предпросмотра, так как производится только один вывод в консоль (по сравнению с 40-50 выводами до этой оптимизации).

#### 5.4.2 Системно зависимые символы

При считывании файла в режиме текста («r») стандартная библиотека определяет операционную систему и заменяет системно зависимые символы исходного файла на соответствующие для программиста. Одним из таких

символов является символ новой строки. Например, если файл редактировался под операционной системой Windows, то этот символ будет представлять собой два байта – \n\r. Однако при считывании файла операционная система вернет только один байт – \n, из-за этого размер файлов и не совпадает. Если при считывании текста такая подстановка только облегчает работу с вводом, то при попытке вывести каждый байт (особенно если считываемый файл даже не является текстовым), то появляется разница в размере, а также внутренним представлении файла. Данная проблема решается простой заменой режима считывания на бинарный – «rb».

## 5.5 Просмотр дисков

Перемещение по директориям осуществляется при помощи относительных ссылок. Так, чтобы перейти в директорию выше используется ссылка «..». Однако такой подход является не совсем корректным, так как при попытке перейти выше, находясь в корневом каталоге, то библиотека вернет тот же самый путь, то есть не поменяет директорию.

Библиотека не предоставляет возможности просматривать доступные диски, а чтобы не менять внешний принцип работы класса Filedirectory (обрабатывающий работу с директориями, в том числе и перемещение по ним), пришлось идти на некоторые уступки в оптимизации кода (однако добавление одной проверки практически никак не влияет на скорость работы ПС).

Первым делом была добавлена следующая проверка:

```
1. auto prev_path = fs::current_path();  
2. fs::current_path(path);  
3. auto current_path = fs::current_path();  
4. if (path == ".." && prev_path == current_path) ...
```

которая семантически означает – «Если пользователь пытается перейти на директорию выше, но директория не меняется, следовательно ПС находится в корневом каталоге диска. Тогда необходимо:».

Далее с помощью запроса WinApi ПС получает битовую маску, которая показывает, существует ли логический диск. Самый младший бит указывает на диск А, следующий на Б и т.д. Опрос происходит по следующему алгоритму:

```
1. DWORD drives = GetLogicalDrives();  
2. for (int drive = 1, letter = 'A'; drive != 0x80000000; drive <= 1, letter++)  
3.     if ((drives & drive) != 0) ...
```



Следовательно заполняется массив с файлами, название которых состоит из слов «Logical Drive» и буквы диска. Таким образом внешняя работа класса осталась неизменной, запросы на него выполняются точно также, однако теперь отображаются возможные диски.

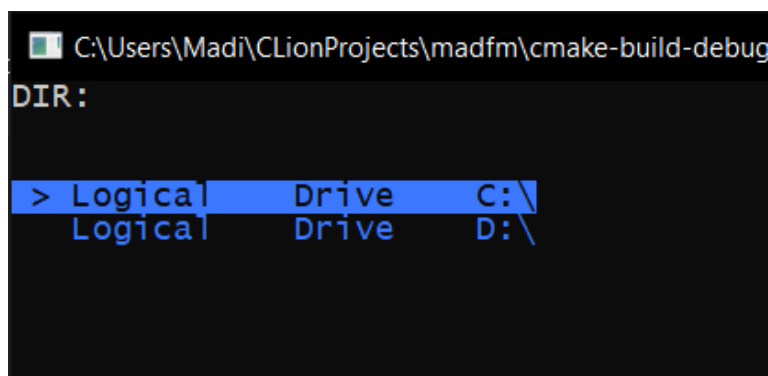


Рисунок 5.2 – Диски

## 5.6 Вывод из прохождения тестирования

После прохождения тестирования в ПС были найдены критические ошибки. После их устранения основные возможности файлового менеджера стали доступными для пользователя, а их функциональность подтверждена, как работающая. Возможно существование других, более глубоких ошибок, однако они не были выявлены, при нормальной работе операционной системы, а обработка ошибок гарантирует, что даже при экстренных ситуациях, информация пользователя не будет повреждена.

## 6 РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ

Скачать исполняемый файл можно из любого доступного источника. Программа готова к использованию.

### 6.1 Предустановки

Для более удобного использования файлового менеджера рекомендуется настроить переменную PATH.

1) Необходимо ввести System variables в меню поиска пуск и нажать Enter:

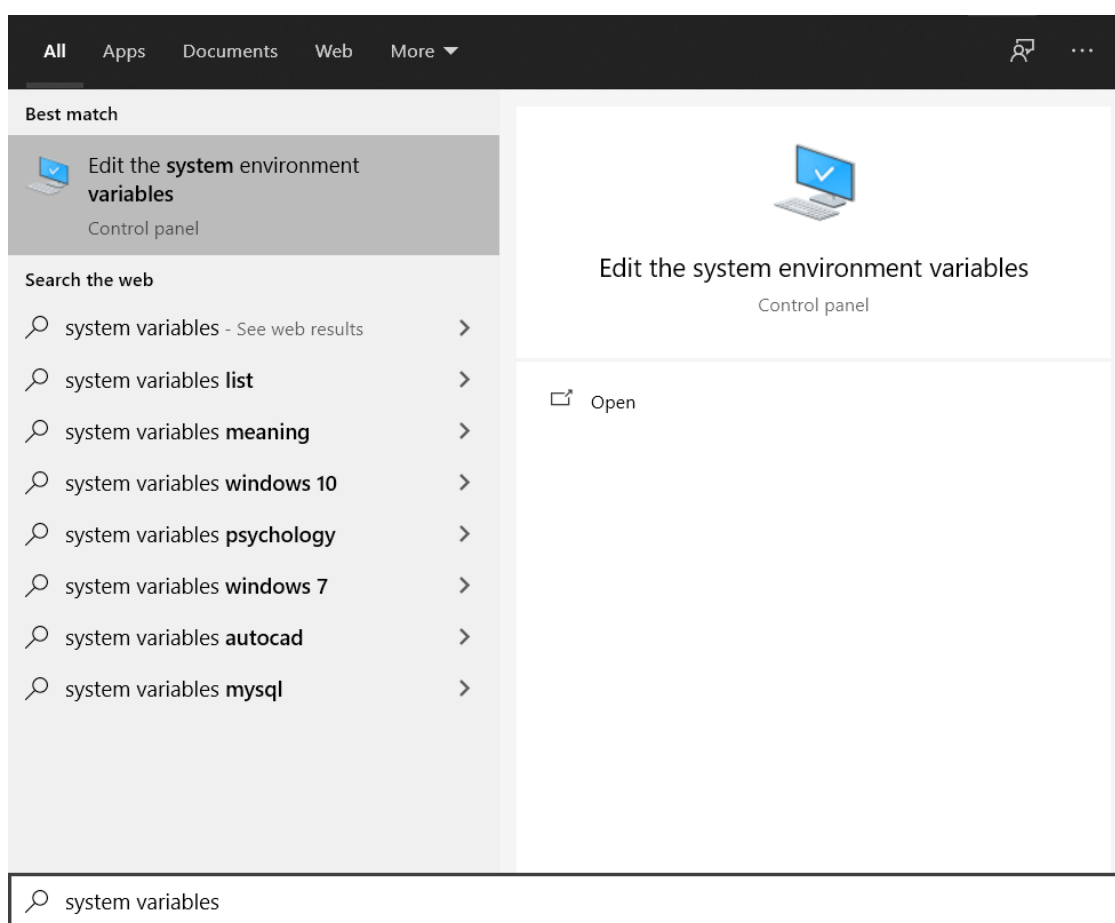


Рисунок 6.1 – Меню пуск

2) Далее необходимо перейти в меню Environment variables:

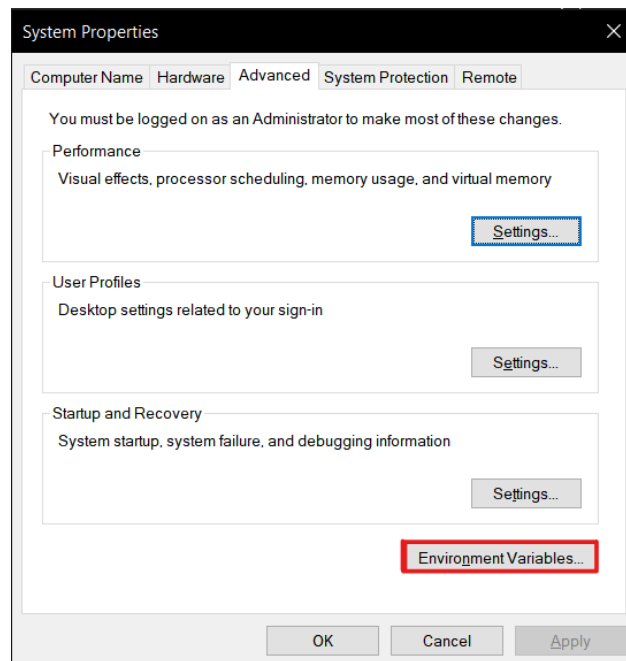


Рисунок 6.2 – Переменные среды

3) В разделе System variables необходимо два раза нажать по полю PATH:

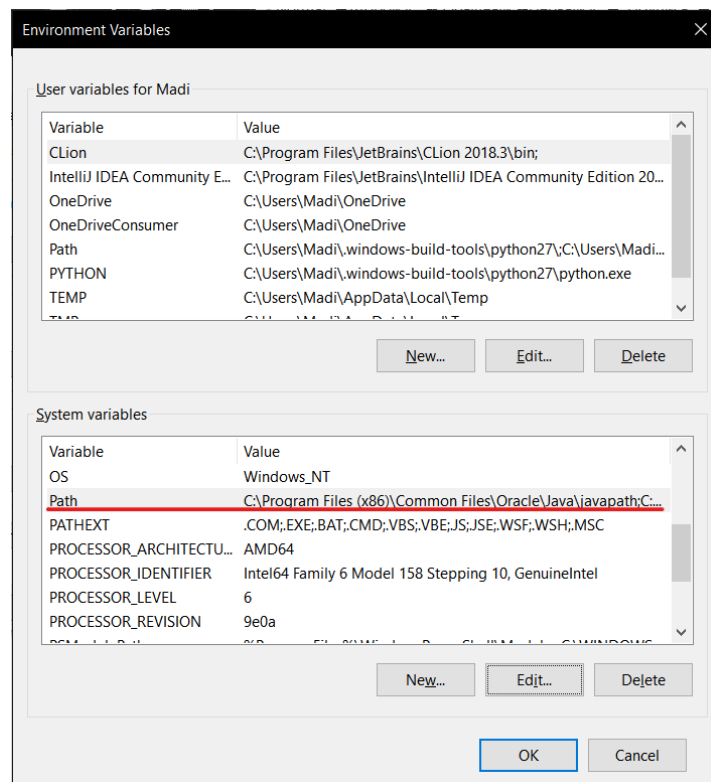


Рисунок 6.3 – Свойство пути

4) В появившемся окне необходимо создать новое правило, путем нажатия на кнопку New:

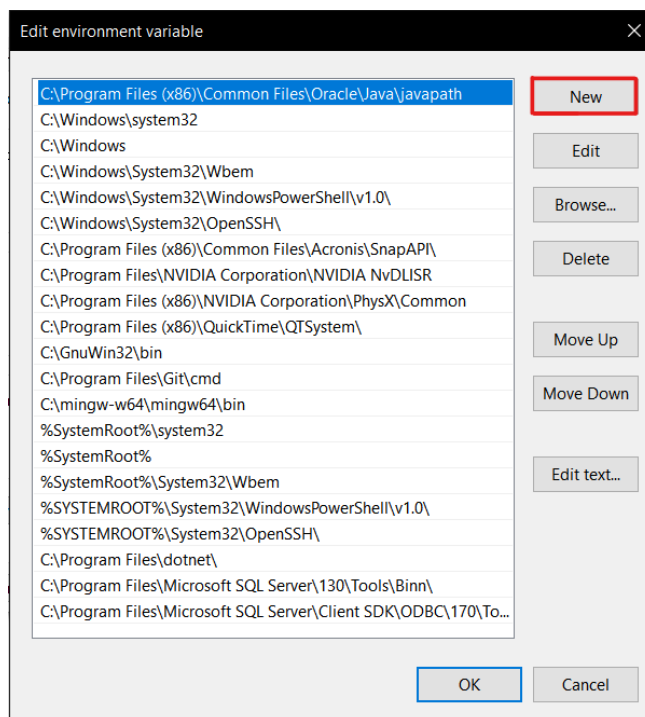


Рисунок 6.4 – Новое свойство

5) Далее следует ввести полный путь до папки, содержащую в себе программу с файловым менеджером:

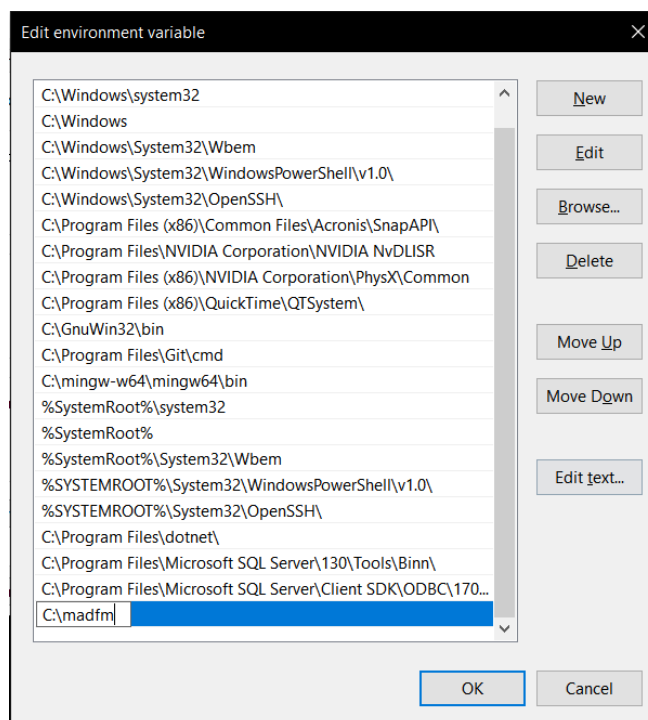


Рисунок 6.5 – Путь до программы

6) После установки свойства необходимо закрыть все окна, нажимая на кнопку ОК.

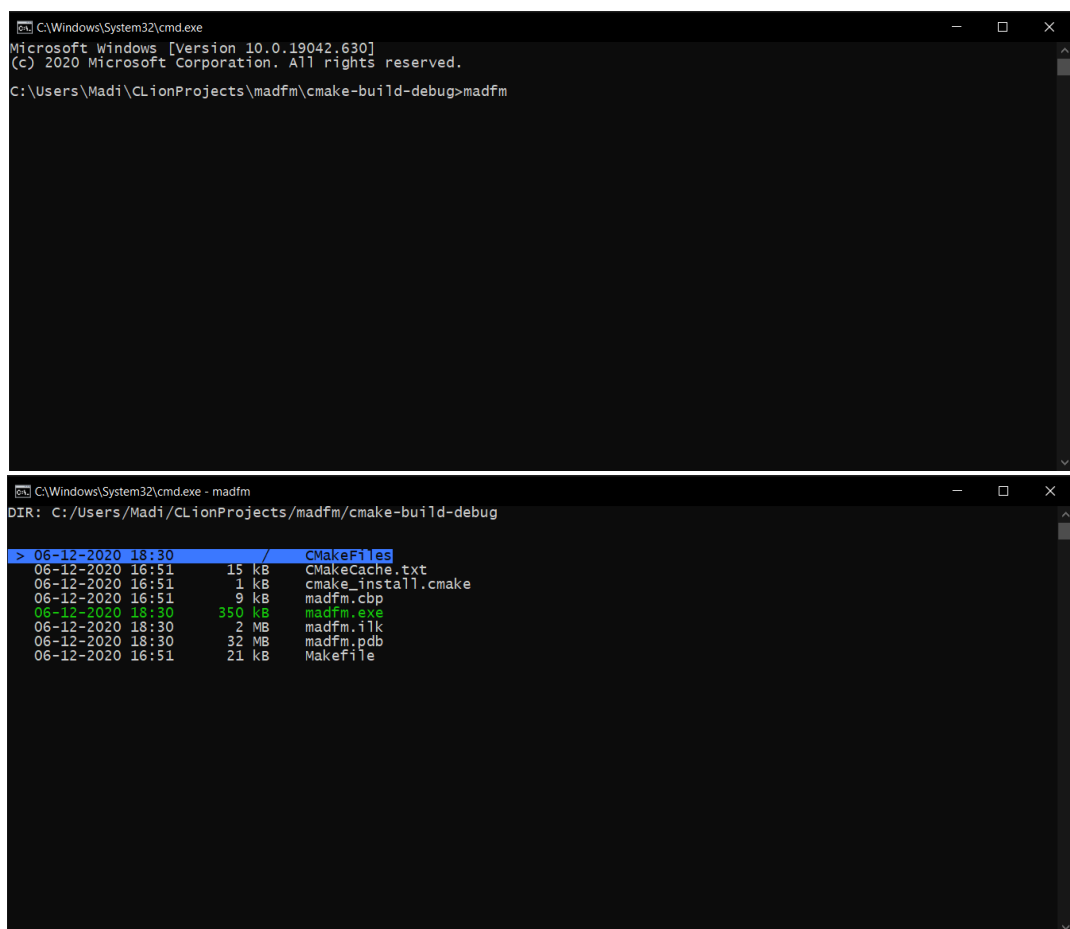
После выполнения данных действий запускать файловый менеджер можно будет из любого места файловой системы.

## 6.2 Запуск файлового менеджера

Для того чтобы запустить файловый менеджер существует 2 способа.

### 6.2.1 Запуск через командную строку

Находясь в запущенной сессии командной строки необходимо ввести `madfm` и нажать Enter.



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19042.630]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\Madi\CLionProjects\madfm\cmake-build-debug>madfm

C:\Windows\System32\cmd.exe - madfm
DIR: C:/Users/Madi/CLionProjects/madfm/cmake-build-debug
> 06-12-2020 18:30 / CMakeFiles
06-12-2020 16:51 15 kB CMakeCache.txt
06-12-2020 16:51 1 kB cmake_install.cmake
06-12-2020 16:51 9 kB madfm.cbp
06-12-2020 18:30 350 kB madfm.exe
06-12-2020 18:30 2 MB madfm.ilc
06-12-2020 18:30 32 MB madfm.pdb
06-12-2020 16:51 21 kB Makefile
```

Рисунок 6.6 – Запуск в консоли

### 6.2.2 Запуск через программу Run

1) Необходимо нажать сочетание клавиш Win+R. Откроется следующее окно:

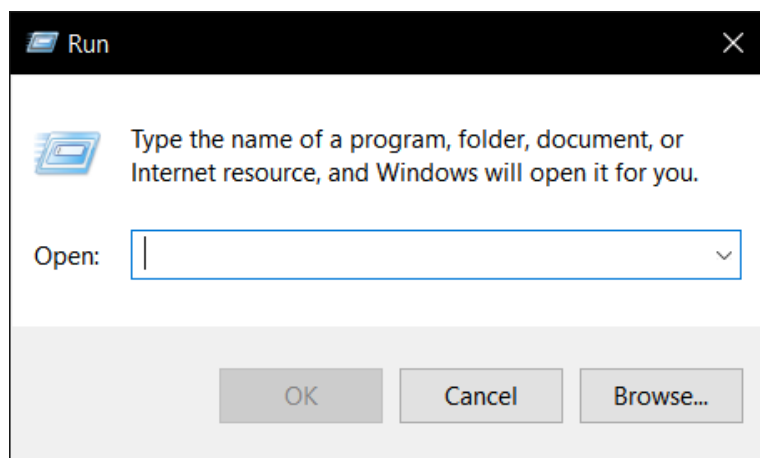


Рисунок 6.7 – Программа Run

2) Далее следует ввести madfm и нажать Enter:

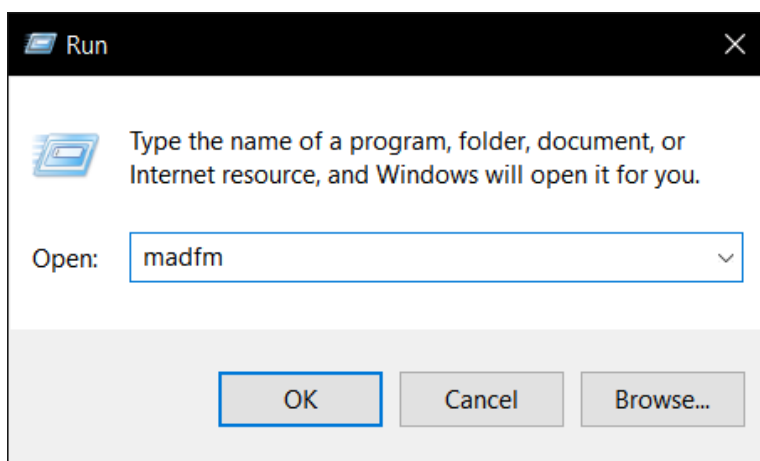


Рисунок 6.8 – Запуск через программу Run

От выбора способа запуска зависит изначальная рабочая директория файлового менеджера – при запуске через сессию терминала рабочая директория будет та, что была рабочей директорией терминала; при запуске через программу Run – корневой каталог пользователя (в моем случае C:\Users\Madi).

## 6.3 Использование файлового менеджера

Данный файловый менеджер включает в себя 8 функций – перемещение по директориям, открытие файла, переименование, создание, удаление, предпросмотр, перемещение и копирование.

Нет необходимости запоминать все сочетания клавиш – для этого всегда можно воспользоваться меню помощи на клавишу F2.

### 6.3.1 Навигация

Меню файлового менеджера разделено на 6 условных участков:

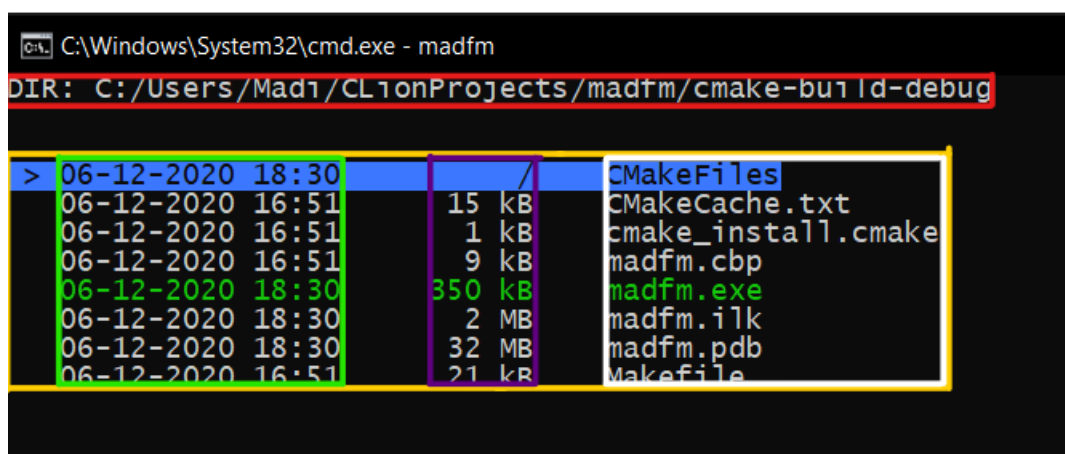


Рисунок 6.9 – Меню программы

Рабочая директория (красный) – обозначает, какая директория открыта и отображена в данный момент.

Таблица файлов (желтый) – список всех файлов и папок, содержащиеся в рабочей директории.

Столбец последнего редактирования (зеленый) – время и дата последнего редактирования файла или папки.

Столбец памяти (фиолетовый) – занимаемое место на диске (для файлов). Память для папок не рассчитывается, так как это занимает значительное время и количество обработок процессором.

Столбец названий (белый) – название файла или папки.

Выделенный элемент (голубой) – элемент, над которым будут происходить все последующие действия.

Для удобства в таблице разные типы файлов отображаются разными цветами. Синим цветом отображается директория, зеленым – исполняемый файл.

### 6.3.2 Перемещение по директориям

Перемещение по директориям происходит с помощью стрелок или клавиш **h j k l**. Клавиши **h j k l** переводятся в управление клавишами влево, вниз, вверх, вправо, соответственно.

- 1) Клавиша влево меняет директорию на родительскую.
- 2) Клавиша вправо меняет директорию на выделенную. Если нажать вправо, при выделенной не директории, ничего не произойдет.
- 3) Клавиша вверх меняет выделенный файл на предыдущий. Если выделен самый первый файл, выделение «обернется» и станет самым последним.
- 4) Клавиша вниз меняет выделенный файл на следующий. Если выделен самый последний файл, выделение «обернется» и станет самым первым.

### 6.3.3 Открытие файла или папки

Открытие файла или папки происходит путем нажатия на клавишу **O**. Данное действие:

- 1) Меняет рабочую директорию (если выделенной была директория).
- 2) Открывает файл в программе по умолчанию.

### 6.3.4 Переименование файла или папки

Переименование файла или папки происходит путем нажатия на клавишу **R**. Данное действие запросит новое название для элемента (которое ограничено 64 символами). Для подтверждения операции необходимо нажать **Enter**. Для отмены операции следует нажать **Esc**.

### 6.3.5 Создание файла или папки

Создание нового элемента происходит путем нажатия на клавишу **N**. Данное действие запросит название для нового элемента (которое ограничено 64 символами). Для подтверждения операции необходимо нажать **Enter**. После подтверждения необходимо:

- 1) Нажать клавишу **F** для создания файла.
- 2) Нажать клавишу **D** для создания директории.
- 3) Нажать клавишу **Esc** для отмены операции.

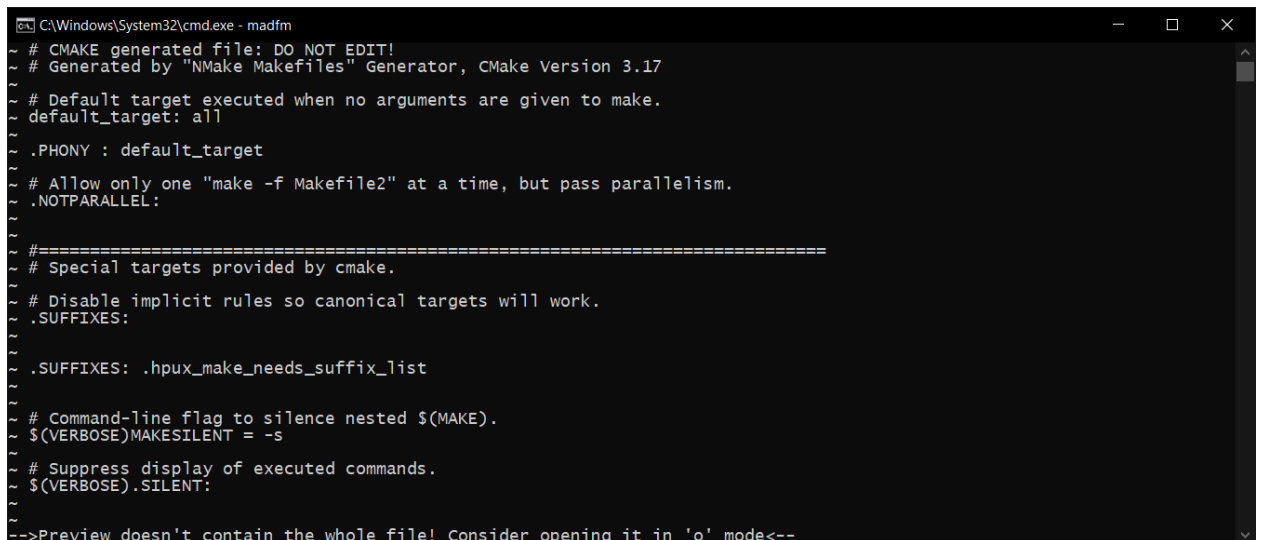
### 6.3.6 Удаление папки или файла

Удаление нового элемента происходит путем нажатия на клавишу **D**. Данное действие запросит подтверждение удаления. Подтвердить удаление возможно только путем нажатия на клавишу **Y**. При нажатии любой другой клавиши операция отменяется. **Внимание!** Данная операция необратима, следует быть с ней осторожнее.



### 6.3.7 Предпросмотр

Предпросмотр элемента происходит путем нажатия на клавишу V, Shift+V. Данное действие покажет максимальное количество информации, которое можно уместить на консоль или весь файл (при малом его размере). При нажатии на клавишу V, выводится текстовый предпросмотр:

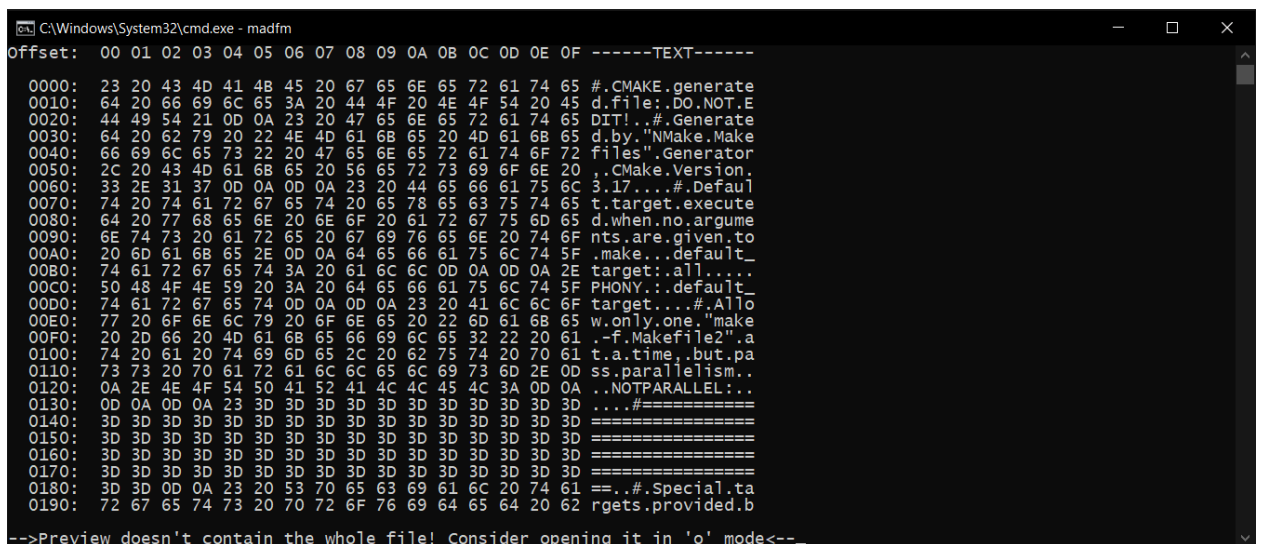


```
C:\Windows\System32\cmd.exe - madfm
~ # CMAKE generated file: DO NOT EDIT!
~ # Generated by "NMake Makefiles" Generator, CMake Version 3.17
~
~ # Default target executed when no arguments are given to make.
~ default_target: all
~
~ .PHONY : default_target
~
~ # Allow only one "make -f Makefile2" at a time, but pass parallelism.
~ .NOTPARALLEL:
~
~ =====
~ # Special targets provided by cmake.
~
~ # Disable implicit rules so canonical targets will work.
~ .SUFFIXES:
~
~ .SUFFIXES: .hpux_make_needs_suffix_list
~
~ # Command-line flag to silence nested $(MAKE).
~ $(VERBOSE)MAKESILENT = -s
~
~ # Suppress display of executed commands.
~ $(VERBOSE).SILENT:
~
-->Preview doesn't contain the whole file! Consider opening it in 'o' mode--
```

Рисунок 6.10 – Текстовый предпросмотр

где тильда (~) означает начало каждой строки. Если файл полностью не помещается на экран, снизу будет выведено соответствующее сообщение.

При нажатии сочетания клавиш Shift+V на экран выводится содержимое файла в шестнадцатеричном формате:



```
C:\Windows\System32\cmd.exe - madfm
Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F -----TEXT-----
0000: 23 20 43 4D 41 48 45 20 67 65 6E 65 72 61 74 65 #.CMAKE.generate
0010: 64 20 68 69 6C 65 3A 20 44 4F 20 4E 4F 54 20 45 d.file: DO.NOT.E
0020: 44 49 54 21 0D 0A 23 20 47 65 6E 65 72 61 74 65 DIT!.,#.Generate
0030: 64 20 62 79 20 22 4E 4D 61 68 65 20 4D 61 68 65 d.by:"NMake.Make
0040: 66 69 6C 65 73 22 20 47 65 6E 65 72 61 74 6F 72 files".Generator
0050: 2C 20 43 4D 61 68 65 20 56 65 72 73 69 6F 6E 20 ..CMake.Version.
0060: 33 2E 31 37 0D 0A 0D 0A 23 20 44 65 66 61 75 6C 3.17....#.Defaul
0070: 74 20 74 61 72 67 65 74 20 65 78 65 63 75 74 65 t.target.execute
0080: 64 20 77 68 65 6E 20 6E 6F 20 61 72 67 75 6D 65 d.when.no,argume
0090: 6E 74 73 20 61 72 65 20 67 69 76 65 6E 20 74 6F nts,are.given.to
00A0: 20 6D 61 68 65 2E 0D 0A 64 65 66 61 75 6C 74 5F .make...default_
00B0: 74 61 72 67 65 74 3A 20 61 6C 6C 0D 0A 0D 0A 2E target:.all....
00C0: 50 48 4F 4E 59 20 3A 20 64 65 66 61 75 6C 74 5F PHONY:..default_
00D0: 74 61 72 67 65 74 0D 0A 0D 0A 23 20 41 6C 6C 6F target....#.Allo
00E0: 77 20 6F 6E 6C 79 20 6F 6E 65 20 22 6D 61 68 65 w.only.one."make
00F0: 20 2D 66 20 4D 61 68 65 66 69 6C 65 32 22 20 61 .-f.Makefile2".a
0100: 74 20 61 20 74 69 6D 65 2C 20 62 75 74 20 70 61 t.a.time,.but.pa
0110: 73 73 20 70 61 72 61 6C 6C 65 6C 69 73 6D 2E 0D ss.parallelism..
0120: 0A 2E 4E 4F 54 50 41 52 41 4C 4C 45 4C 3A 0D 0A ..NOTPARALLEL:...
0130: 0D 0A 0D 0A 23 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D ....#=====
0140: 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D =====
0150: 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D =====
0160: 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D =====
0170: 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D =====
0180: 3D 3D 0D 0A 23 20 53 70 65 63 69 61 6C 20 74 61 ==.,#.Special.ta
0190: 72 67 65 74 73 20 70 72 6F 76 69 64 65 64 20 62 rgets.provided.b
-->Preview doesn't contain the whole file! Consider opening it in 'o' mode--
```

Рисунок 6.11 – Hexdump

где правая колонка показывает содержимое в формате кодировки ASCII.

### 6.3.8 Перемещение и копирование

Первым делом для перемещения и копирования файла или папки необходимо выбрать источник, путем нажатия клавиши С. Далее следует поменять рабочую и директорию и нажать:

- 1) М – для перемещения файла или папки (источник удаляется).
- 2) Р – для копирования файла или папки (источник не меняется).

### 6.4 Частые ошибки

Ниже приведены наиболее часто возникающие ошибки и способы их исправления. При возникновении какого либо сообщения об ошибке следует найти ее в этой таблице и исправить так, как описано в колонке «Исправление».

Таблица 6.1 – Ошибки в программе

№	Сообщение	Причина	Исправление
1	Windows cannot find 'madfm'. Make sure you typed the name correctly, and then try again.	Не установлена системная переменная PATH.	Установка системной переменной детально описана в секции «6.1 Предустановки».
2	'madfm' is not recognized as an internal or external command, operable program or batch file.		
3	Access is denied!	Отсутствует доступ на редактирование.	Следует запустить программу с правами администратора. <b>Внимание!</b> Данное действие может повлечь за собой необратимые последствия. Рекомендуется работать в директориях, для которых не нужны права администратора.

Продолжение таблицы 6.1

№	Сообщение	Причина	Исправление
4	File not found!	Программа не может найти запрошенный файл. Возможно он был удален или перемещен извне.	Следует перезапустить программу.
5	Preserved name!	Данное имя не может быть использовано как имя пользовательского файла.	Следует выбрать другое название для папки или файла.
6	File or directory with such name already exists!	Рабочая директория содержит элемент с таким названием.	
7	Preview doesn't contain the whole file!	Содержимое файла не помещается в режим предпросмотра.	Следует открыть файл в другой программе.

## ЗАКЛЮЧЕНИЕ

В настоящее время популярность консольных файловых менеджеров заметно падает. На их замену пришли быстрые и удобные графические интерфейсы, с возможностью переноса предметов с помощью мыши, открытия нескольких сессий и т.д. Однако время от времени появляется необходимость использования консольного приложения, которое отличается от графических тем, что не нужно отрывать руки от клавиатуры. Тем не менее, самым главным достоинством консольного приложения является его независимость от среды рабочего стола, что может пригодиться при удаленном подключении.

В рамках данного курсового проекта было реализовано консольное программное средство `madfm`, которое обеспечит быструю и удобную интеграцию файлового менеджера в среду консольного приложения. Согласно поставленным задачам, в данном приложении были реализованы следующие функции:

- a) графическая визуализация пунктов меню (содержимого директории);
- b) перемещение по директориям;
- c) создание директорий и файлов;
- d) удаление директорий и файлов;
- e) копирование директорий и файлов;
- f) переименование директорий и файлов;
- g) предпросмотр файлов;
- h) отображение меню помощи с управлением;
- i) индикация ошибок.

Для успешного выполнения всех поставленных целей потребовалось изучить объектно-ориентированные возможности языка C++, изучить основные начальные принципы данной парадигмы, а также изучить возможности основного набора базовых функций интерфейсов программирования приложений операционной системы Windows.

В дальнейшем возможны улучшения и доработки, такие как платформо-независимость, с сохранением скорости работы и простоты использования, возможность отменять и возвращать действия (`undo`, `redo`), поддержка различных соотношений сторон, поддержка шрифтов и внутренний редактор текстов.

Использование данного приложения позволит ускорить работу с файловой системой, при отсутствии среды рабочего стола, при этом сохраняя все удобство и привычные, для большинства, сочетания клавиш. А благодаря использованию `WinApi` остается приятное впечатление от скорости работы приложения, ведь оно не тормозит, даже при большом количестве файлов.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] cppreference [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://en.cppreference.com/w/cpp/filesystem>
- [2] Wikipedia [Электронный ресурс]. — Свободная энциклопедия. — Режим доступа: [https://en.wikipedia.org/wiki/File\\_manager](https://en.wikipedia.org/wiki/File_manager)
- [3] Github [Электронный ресурс]. — Веб-сервис. — Режим доступа: <https://github.com/AntonTymoshchuk/FileManager>
- [4] Github [Электронный ресурс]. — Веб-сервис. — Режим доступа: <https://github.com/wwwIgorNet/File-Manager>
- [5] Github [Электронный ресурс]. — Веб-сервис. — Режим доступа: <https://github.com/vanyscore/FileManager>
- [6] Бахтизин, В. В. Конструирование программ и языки программирования : учебно-метод. пособие для студентов специальности «Програм. обеспечение информац. технологий» : в 2 ч. Ч. 1 : Язык Си / В. В. Бахтизин, И. М. Марина, Е. В. Шостак. — Минск : БГУИР, 2006. — 48 с.

## ПРИЛОЖЕНИЕ А

### Исходный код программы

#### Класс File

```
1. #include "../headers/files/File.h"
2. #include "../headers/files/FileDirectoryUtils.h"
3.
4. File::File(const std::filesystem::directory_entry &entry) {
5.     name_orig = entry.path().filename().string();
6.     name_parsed = FileDirectoryUtils::parseName(name_orig);
7.     type = FileDirectoryUtils::defineType(entry);
8.     size = FileDirectoryUtils::parseSize(entry.file_size(), type == FileType::DIR);
9.     time = FileDirectoryUtils::parseTime(entry.last_write_time());
10. }
11.
12. File::File(const std::string &drive) {
13.     name_orig = drive;
14.     name_parsed = name_orig;
15.     type = FileType::DIR;
16.     size = "Drive";
17.     time = "Logical";
18. }
19.
20. std::string File::getName() {
21.     return name_orig;
22. }
23.
24. std::string File::getParsedName() {
25.     return name_parsed;
26. }
27.
28. FileType File::getType() {
29.     return type;
30. }
31.
32. std::string File::getSize() {
33.     return size;
34. }
35.
36. std::string File::getTime() {
37.     return time;
38. }
```

#### Класс FileDirectory

```
1. #include "../headers/files/FileDirectory.h"
2. #include "../headers/files/FileDirectoryUtils.h"
3. #include <string>
4. #include <filesystem>
5. #include <fstream>
6.
7. #include <windows.h>
8.
9. namespace fs = std::filesystem;
10.
11. FileDirectory::FileDirectory(const std::string &path) {
12.     reInit(path);
13.     is_drives = false;
14. }
15.
```

```

16. FileDirectoryException FileDirectory::reInit(const std::string &path) {
17.     std::error_code ec;
18.     const auto& entry = fs::directory_iterator(path, ec);
19.     if (ec.value() != 0)
20.         return FileDirectoryUtils::handleExceptionCode(ec.value());
21.     fillList(path);
22.     return FileDirectoryException::NO_EXCEPTION;
23. }
24.
25. void FileDirectory::fillList(const std::string& path) {
26.     auto prev_path = fs::current_path();
27.     fs::current_path(path);
28.     auto current_path = fs::current_path();
29.     if (path == ".." && prev_path == current_path && !is_drives) {
30.         is_drives = true;
31.         fillListDrives();
32.         return;
33.     }
34.     is_drives = false;
35.     list.clear();
36.     int last_dir_index = 0;
37.     for (const auto& entry : fs::directory_iterator(current_path)) {
38.         File f(entry);
39.         if (f.getType() == FileType::DIR) {
40.             auto it = list.begin() + last_dir_index;
41.             list.insert(it, f);
42.             last_dir_index++;
43.         } else {
44.             list.push_back(f);
45.         }
46.     }
47. }
48.
49. void FileDirectory::fillListDrives() {
50.     list.clear();
51.     DWORD drives = GetLogicalDrives();
52.     for (unsigned int drive = 1, letter = 'A'; drive != 0x80000000; drive <<= 1, letter++)
53.     {
54.         if ((drives & drive) != 0) {
55.             std::string drive_name = std::string(1, letter);
56.             drive_name.append(":/");
57.             File f(drive_name);
58.             list.push_back(f);
59.         }
60.     }
61.
62. std::vector<File> FileDirectory::getFilesList() {
63.     return list;
64. }
65.
66. std::string FileDirectory::getCurrentDirectory() const {
67.     if (is_drives) {
68.         return "";
69.     }
70.     return fs::current_path().generic_string();
71. }
72.
73. FileDirectoryException FileDirectory::move(const std::string &old_path, const std::string
    &new_path) {
74.     std::error_code ec;
75.     fs::rename(old_path, new_path, ec);
76.     if (ec.value() != 0)
77.         return FileDirectoryUtils::handleExceptionCode(ec.value());
78.     return FileDirectoryException::NO_EXCEPTION;
79. }
80.

```

```

81. FileDirectoryException FileDirectory::copy(const std::string &old_path, const std::string
    &new_path) {
82.     std::error_code ec;
83.     fs::copy(old_path, new_path, ec);
84.     if(ec.value() != 0)
85.         return FileDirectoryUtils::handleExceptionCode(ec.value());
86.     return FileDirectoryException::NO_EXCEPTION;
87. }
88.
89. FileDirectoryException FileDirectory::changeName(const std::string& old_name, const std::s
    tring& new_name) {
90.     auto old_p = fs::current_path() / old_name;
91.     auto new_p = fs::current_path() / new_name;
92.     return move(old_p.string(), new_p.string());
93. }
94.
95. FileDirectoryException FileDirectory::createDir(const std::string &name) {
96.     std::error_code ec;
97.     auto path = fs::current_path() / name;
98.     fs::create_directory(path, ec);
99.     if (ec.value() != 0)
100.        return FileDirectoryUtils::handleExceptionCode(ec.value());
101.    return FileDirectoryException::NO_EXCEPTION;
102. }
103.
104. FileDirectoryException FileDirectory::createFile(const std::string &name) {
105.     auto path = fs::current_path() / name;
106.     std::ofstream file;
107.     file.open(path.string());
108.     file.close();
109.     if(errno != 0)
110.        return FileDirectoryUtils::handleExceptionCode(errno);
111.    return FileDirectoryException::NO_EXCEPTION;
112. }
113.
114. bool FileDirectory::containsCurrent(const std::string &path) {
115.     return fs::exists(fs::current_path() / path);
116. }
117.
118. FileDirectoryException FileDirectory::deleteFile(const std::string &name) {
119.     std::error_code ec;
120.     auto path = fs::current_path() / name;
121.     fs::remove_all(path, ec);
122.     if (ec.value() != 0)
123.        return FileDirectoryUtils::handleExceptionCode(ec.value());
124.    return FileDirectoryException::NO_EXCEPTION;
125. }

```

## Класс FileDirectoryUtils

```

1. #include "../headers/files/FileDirectoryUtils.h"
2.
3. std::string FileDirectoryUtils::parseName(std::string name) {
4.     if (name.length() <= MAX_CHARS)
5.         return name;
6.
7.     name = name.substr(0, MAX_CHARS - TOO_LONG_POSTFIX.length());
8.     name.append(TOO_LONG_POSTFIX);
9.     return name;
10. }
11.
12. std::string FileDirectoryUtils::parseTime(std::filesystem::file_time_type ftime) {
13.     using namespace std::chrono;
14.     using namespace std::filesystem;
15.

```



```

16.     auto sctp = time_point_cast<system_clock::duration>(ftime - file_time_type::clock::now
17.     ())
18.                                     + system_clock::now());
19.     time_t ttime = system_clock::to_time_t(sctp);
20.     char buff[17];
21.     tm current_time;
22.     localtime_s(&ttime, &current_time);
23.     strftime(buff, 17, TIME_FORMAT.c_str(), &ttime);
24.
25.     return buff;
26. }
27.
28. std::string FileDirectoryUtils::parseSize(uintmax_t size, bool is_dir) {
29.     char directory_size[SIZE_LENGTH + 1] = {0};
30.     if (is_dir) {
31.         sprintf_s(directory_size, SIZE_LENGTH + 1, "%s", SIZE_LENGTH, "/");
32.         return std::string(directory_size);
33.     }
34.
35.     int index = 0;
36.     while (size > 1024) {
37.         size >>= 10;
38.         index++;
39.     }
40.     std::string size_string = std::to_string(size).append(" ").append(UNITS[index]);
41.     sprintf_s(directory_size, SIZE_LENGTH + 1, "%s", SIZE_LENGTH, size_string.c_str());
42.     return std::string(directory_size);
43. }
44.
45. FileType FileDirectoryUtils::defineType(const std::filesystem::directory_entry &entry) {
46.     if(entry.is_directory())
47.         return FileType::DIR;
48.     if(isFileExe(entry.path().filename().string()))
49.         return FileType::EXE;
50.     return FileType::ORD;
51. }
52.
53. bool FileDirectoryUtils::isFileExe(const std::string &name) {
54.     std::string ext = defineExtension(name);
55.     return ext == "exe" || ext == "lnk";
56. }
57.
58. std::string FileDirectoryUtils::defineExtension(const std::string &name) {
59.     return name.substr(name.find_last_of('.') + 1);
60. }
61.
62. FileDirectoryException FileDirectoryUtils::handleExceptionCode(int code) {
63.     switch (code) {
64.         case 13:
65.         case 5:
66.             return FileDirectoryException::ACCESS_DENIED;
67.         case 3:
68.             return FileDirectoryException::FILE_NOT_FOUND;
69.         case 2:
70.         case 267:
71.         case 183:
72.             return FileDirectoryException::INCORRECT_NAME;
73.         default:
74.             return FileDirectoryException::UNHANDLED;
75.     }
76. }

```

## Класс FileCreator

```
1. #include <conio.h>
2. #include "../headers/gui/modules/FileCreator.h"
3.
4. FileCreator::FileCreator(ConsoleGuiHandler *cgh) : cgh(cgh) {}
5.
6. void FileCreator::createFileOrDir() {
7.     cgh->redrawConsoleGui();
8.     cgh->utils.outputLine(ConsoleGuiHandler::NAME_QUESTION, cgh->saved_attributes);
9.     std::string name = cgh->utils.inputLine(cgh->saved_attributes);
10.    cgh->redrawConsoleGui();
11.    if(!cgh->checkFile(name)) return;
12.    cgh->utils.outputLine(FD_QUESTION, cgh->saved_attributes);
13.    int c;
14.    iloop:
15.        c = getch();
16.        switch(c) {
17.            case 'd':
18.            case 'D':
19.                createDir(name);
20.                break;
21.            case 'f':
22.            case 'F':
23.                createFile(name);
24.                break;
25.            case 27:
26.                cgh->redrawConsoleGui();
27.                break;
28.            default:
29.                goto iloop;
30.        }
31. }
32.
33. void FileCreator::createDir(const std::string &name) {
34.     FiledirectoryException e = Filedirectory::createDir(name);
35.     if(e == FiledirectoryException::NO_EXCEPTION) {
36.         cgh->redrawConsoleGui();
37.         cgh->reInit(".");
38.         cgh->utils.outputLine(SUCCESS_CREATE_DIR, cgh->saved_attributes);
39.     } else {
40.         cgh->outputCorrespondingException(e);
41.     }
42. }
43.
44. void FileCreator::createFile(const std::string &name) {
45.     FiledirectoryException e = Filedirectory::createFile(name);
46.     if(e == FiledirectoryException::NO_EXCEPTION) {
47.         cgh->redrawConsoleGui();
48.         cgh->reInit(".");
49.         cgh->utils.outputLine(SUCCESS_CREATE_FILE, cgh->saved_attributes);
50.     } else {
51.         cgh->outputCorrespondingException(e);
52.     }
53. }
```

## Класс FileDeleter

```
1. #include <conio.h>
2. #include "../headers/gui/modules/FileDeleter.h"
3.
4. FileDeleter::FileDeleter(ConsoleGuiHandler *cgh): cgh(cgh) {}
5.
6. void FileDeleter::deleteFile() {
```

```

7.     cgh->utils.outputLine(DEL_QUESTION, cgh->saved_attributes);
8.     int c;
9.     c = getch();
10.    if(c == 'y' || c == 'Y') {
11.        deleteFile(cgh->list_files[cgh->current_selected_index].getName());
12.    } else {
13.        cgh->redrawConsoleGui();
14.    }
15. }
16.
17. void FileDeleter::deleteFile(const std::string &name) {
18.     FiledirectoryException e = Filedirectory::deleteFile(name);
19.     if(e == FiledirectoryException::NO_EXCEPTION) {
20.         cgh->redrawConsoleGui();
21.         cgh->reInit(".");
22.         cgh->utils.outputLine(SUCCESS_DELETE, cgh->saved_attributes);
23.     } else {
24.         cgh->outputCorrespondingException(e);
25.     }
26. }

```

## Класс FileMover

```

1. #include "../../headers/gui/modules/FileMover.h"
2.
3. FileMover::FileMover(ConsoleGuiHandler *cgh): cgh(cgh), saved_location(""), saved_name("")
4. {
5. }
6.
7. void FileMover::saveLocation() {
8.     saved_name = cgh->list_files[cgh->current_selected_index].getName();
9.     saved_location = cgh->fd.getCurrentDirectory() + "/" + saved_name;
10.    cgh->redrawConsoleGui();
11.    cgh->utils.outputLine(MSG_LOC_SAVED, cgh->saved_attributes);
12. }
13.
14. void FileMover::moveOrCopyHandle(filedirectoryException (*func)(const std::string&, const
std::string&), const std::string &success_msg) {
15.     if(saved_location.empty()) {
16.         return;
17.     }
18.     std::string old_p = saved_location;
19.     std::string new_p = cgh->fd.getCurrentDirectory() + "/" + saved_name;
20.     if(!cgh->checkFile(new_p)) return;
21.     saved_location.clear();
22.     saved_name.clear();
23.     FiledirectoryException e = func(old_p, new_p);
24.     if(e == FiledirectoryException::NO_EXCEPTION) {
25.         cgh->redrawConsoleGui();
26.         cgh->reInit(".");
27.         cgh->utils.outputLine(success_msg, cgh->saved_attributes);
28.     } else {
29.         cgh->outputCorrespondingException(e);
30.     }
31. }
32.
33. void FileMover::moveFile() {
34.     moveOrCopyHandle(filedirectory::move, SUCCESS_MOVE);
35. }
36.
37. void FileMover::copyFile() {
38.     moveOrCopyHandle(filedirectory::copy, SUCCESS_COPY);
39. }

```

## Класс FilePreviewer

```
1. #include <conio.h>
2. #include "../headers/gui/modules/FilePreviewer.h"
3.
4. FilePreviewer::FilePreviewer(ConsoleGuiHandler *cgh) : cgh(cgh) {
5.     GetConsoleScreenBufferInfo(cgh->h_console, &csbi);
6.     columns = csbi.srWindow.Right - csbi.srWindow.Left + 1;
7.     rows = csbi.srWindow.Bottom - csbi.srWindow.Top + 1;
8. }
9.
10. void FilePreviewer::waitForClose() {
11.     int c;
12. eloop:
13.     c = getch();
14.     c = toupper(c);
15.     if(c != 'Q') goto eloop;
16. }
17.
18. void FilePreviewer::showTextPreview() {
19.     FILE *f = fopen(cgh->list_files[cgh->current_selected_index].getName().c_str(), "r");
20.     if(!f) {
21.         return;
22.     }
23.
24.     cgh->utils.clearScreen();
25.     int c;
26.     int skip_start = 0;
27.     std::string output;
28.     for(int i = 0; i < rows - 1; i++) {
29.         for(int j = 0; j < columns; j++) {
30.             if(j == 0 && !skip_start) {
31.                 output.push_back('~');
32.                 output.push_back(' ');
33.                 j++;
34.                 continue;
35.             }
36.             c = fgetc(f);
37.             if(c == '\n' || c == EOF) {
38.                 skip_start = 0;
39.                 if(i != rows - 2)
40.                     output.push_back('\n');
41.                 break;
42.             } else if(c == '\t') {
43.                 for(int k = 0; k < 4; k++) {
44.                     output.push_back(' ');
45.                 }
46.                 j+=3;
47.             } else {
48.                 output.push_back(c);
49.                 if(j == columns - 1)
50.                     skip_start = 1;
51.             }
52.         }
53.     }
54.     if(c != EOF)
55.         output.append(PRE_NOTEND);
56.
57.     cgh->utils.outputLineNoNew(output, cgh->saved_attributes);
58.     waitForClose();
59.     fclose(f);
60.     cgh->cleanRedrawConsoleGui();
61. }
62.
63. void FilePreviewer::showRawPreview() {
```

```

64.     FILE *f = fopen(cgh->list_files[cgh-
>current_selected_index].getName().c_str(), "rb");
65.     if(!f) {
66.         return;
67.     }
68.     if(PRE_NOTEND.size() > columns) {
69.         return;
70.     }
71.
72.     cgh->utils.clearScreen();
73.     std::string output;
74.     std::string ascii_line = " ";
75.     output.append(HEX_OFFSET);
76.     output.append("\n\n");
77.
78.     int c;
79.     int off = 0;
80.     char off_str[9] = {0};
81.     char curr_byte[4] = {0};
82.     for(int i = 0; i < rows - 4; i++) {
83.         for(int j = 0; j < 16; j++) {
84.             c = fgetc(f);
85.             if(c == EOF) {
86.                 goto end;
87.             } else {
88.                 if(j == 0) {
89.                     sprintf(off_str, " %04X: ", off);
90.                     off += 16;
91.                     output.append(off_str);
92.                 }
93.                 sprintf(curr_byte, " %02X", c);
94.                 output.append(curr_byte);
95.             }
96.             if(c > 32 && c < 127)
97.                 ascii_line.push_back(c);
98.             else
99.                 ascii_line.push_back('.');
100.        }
101.        output.append(ascii_line);
102.        ascii_line = " ";
103.        output.push_back('\n');
104.    }
105.    if(c != EOF)
106.        output.append(PRE_NOTEND);
107.    end:
108.    cgh->utils.outputLineNoNew(output, cgh->saved_attributes);
109.    waitForClose();
110.    fclose(f);
111.    cgh->cleanRedrawConsoleGui();
112. }

```

## Класс FileRenamer

```

1. #include "../../headers/gui/modules/FileRenamer.h"
2.
3. FileRenamer::FileRenamer(ConsoleGuiHandler *cgh) : cgh(cgh) {}
4.
5. void FileRenamer::rename() {
6.     cgh->redrawConsoleGui();
7.     cgh->utils.outputLine(cgh->NAME_QUESTION, cgh->saved_attributes);
8.     std::string new_name = cgh->utils.inputLine(cgh->saved_attributes);
9.     cgh->redrawConsoleGui();
10.    if(!cgh->checkFile(new_name)) return;
11.    rename(cgh->list_files[cgh->current_selected_index].getName(), new_name);
12. }

```

```

13.
14. void FileRenamer::rename(const std::string &old_name, const std::string &new_name) {
15.     FileDirectoryException e = FileDirectory::changeName(old_name, new_name);
16.     if(e == FileDirectoryException::NO_EXCEPTION) {
17.         cgh->redrawConsoleGui();
18.         cgh->reInit(".");
19.         cgh->utils.outputLine(SUCCESS_RENAME, cgh->saved_attributes);
20.     } else {
21.         cgh->outputCorrespondingException(e);
22.     }
23. }

```

## Класс HelpDrawer

```

1. #include <conio.h>
2. #include "../headers/gui/modules/HelpDrawer.h"
3.
4. HelpDrawer::HelpDrawer(ConsoleGuiHandler *cgh):cgh(cgh) {}
5.
6. void HelpDrawer::showHelp() {
7.     cgh->utils.clearScreen();
8.     for(const auto& line : HELP_LINES) {
9.         cgh->utils.outputLine(line, cgh->saved_attributes);
10.    }
11.    int c;
12.    do {
13.        c = getch();
14.    } while (c != 'q' && c != 'Q');
15.    cgh->cleanRedrawConsoleGui();
16. }

```

## Класс MenuDrawer

```

1. #include "../headers/gui/modules/MenuDrawer.h"
2.
3. MenuDrawer::MenuDrawer(ConsoleGuiHandler *cgh) : cgh(cgh) {
4.     reserveLines();
5. }
6.
7. void MenuDrawer::reserveLines() {
8.     current_lines.reserve(MAX_FILES + MAX_ADDITIONAL_LINES);
9.     for (int i = 0; i < MAX_FILES + MAX_ADDITIONAL_LINES; i++)
10.         current_lines.emplace_back();
11.     next_lines.reserve(MAX_FILES + MAX_ADDITIONAL_LINES);
12. }
13.
14. void MenuDrawer::redrawConsoleGui() {
15.     next_lines.clear();
16.     addDirLine();
17.     addTopBottomLine(cgh->starting_index);
18.     addFilesLines();
19.     addTopBottomLine(cgh->starting_index + MAX_FILES < cgh->list_files.size());
20.     drawDifferentGui();
21. }
22.
23. void MenuDrawer::addDirLine() {
24.     std::string dir = std::string(CONST_PREFIXES[DIR_PREFIX]).append(cgh-
25. >fd.getCurrentDirectory());
26.     ConsoleLine dir_line(dir, cgh->saved_attributes);
27.     next_lines.push_back(dir_line);
28.     next_lines.emplace_back();
29. }
30. void MenuDrawer::addTopBottomLine(bool expression) {

```

```

31.     ConsoleLine line;
32.     if (expression) {
33.         int amount_of_pages = ((int) cgh->list_files.size() / MAX_FILES) + ((cgh-
>list_files.size() % MAX_FILES) != 0);
34.         int current_page = cgh->starting_index / MAX_FILES + 1;
35.         char buff[256] = {0};
36.         sprintf(buff, "%s%s (%d/%d)", CONST_PREFIXES[NOT_SELECTED_PREFIX].c_str(), CONST_P
REFIXES[TOO_MANY_PREFIX].c_str(),
37.             current_page, amount_of_pages);
38.         line.setAttribute(TOO_MANY_ATTR);
39.         line.setText(buff);
40.     }
41.     next_lines.push_back(line);
42. }
43.
44. void MenuDrawer::addSelected(int index) {
45.     ConsoleLine selected_line(CONST_PREFIXES[SELECTED_PREFIX], SELECTED_ATTR);
46.     appendFileInfo(selected_line, index);
47.     next_lines.push_back(selected_line);
48. }
49.
50. void MenuDrawer::addNotSelected(int index) {
51.     ConsoleLine not_selected_line;
52.     switch(cgh->list_files[index].getType()) {
53.         case FileType::DIR:
54.             not_selected_line.setAttribute(DIR_ATTR);
55.             break;
56.         case FileType::EXE:
57.             not_selected_line.setAttribute(EXE_ATTR);
58.             break;
59.         case FileType::ORD:
60.             not_selected_line.setAttribute(cgh->saved_attributes);
61.             break;
62.     }
63.     not_selected_line.setText(CONST_PREFIXES[NOT_SELECTED_PREFIX]);
64.     appendFileInfo(not_selected_line, index);
65.     next_lines.push_back(not_selected_line);
66. }
67.
68. inline void MenuDrawer::appendFileInfo(ConsoleLine& line, int index) {
69.     File f = cgh->list_files[index];
70.     std::string line_text = line.getText().append(f.getTime()).append(CONST_PREFIXES[TAB_P
REFIX]);
71.         .append(f.getSize()).append(CONST_PREFIXES[TAB_PREFIX]).append(f.getParsedName
());
72.     line.setText(line_text);
73. }
74.
75. void MenuDrawer::addFilesLines() {
76.     for (int i = cgh->starting_index; i < MAX_FILES + cgh->starting_index && i < cgh-
>list_files.size(); i++) {
77.         if (i == cgh->current_selected_index)
78.             addSelected(i);
79.         else
80.             addNotSelected(i);
81.     }
82. }
83.
84. void MenuDrawer::cleanRedrawConsoleGui() {
85.     cgh->utils.clearScreen();
86.     current_lines.clear();
87.     reserveLines();
88.     redrawConsoleGui();
89. }
90.
91. void MenuDrawer::drawDifferentGui() {
92.     for (int i = 0; i < next_lines.size(); i++) {

```

```

93.         if (current_lines[i] != next_lines[i]) {
94.             COORD current_line_coord = { 0, (short) i };
95.             SetConsoleCursorPosition(cgh->h_console, current_line_coord);
96.             cgh->utils.clearCurrentLine();
97.             cgh->utils.outputLine(next_lines[i]);
98.             current_lines[i] = next_lines[i];
99.         }
100.     }
101.     reduceCurrentLines();
102. }
103.
104. void MenuDrawer::reduceCurrentLines() {
105.     if (next_lines.size() < current_lines.size()) {
106.         COORD line_coord;
107.         for (int i = next_lines.size(); i < current_lines.size(); i++) {
108.             line_coord.X = 0;
109.             line_coord.Y = i;
110.             cgh->utils.clearLine(line_coord);
111.             current_lines[i] = ConsoleLine();
112.         }
113.         line_coord.X = 0;
114.         line_coord.Y = next_lines.size();
115.         SetConsoleCursorPosition(cgh->h_console, line_coord);
116.     }
117. }

```

## Класс ConsoleGuiHandler

```

1. #include "../headers/gui/ConsoleGuiHandler.h"
2. #include <iostream>
3. #include <utility>
4. #include <conio.h>
5.
6. #include <cstdio>
7.
8. ConsoleGuiHandler::ConsoleGuiHandler(HANDLE h_console) : utils(h_console), h_console(h_console), fd(".") {
9.     CONSOLE_SCREEN_BUFFER_INFO console_info;
10.    GetConsoleScreenBufferInfo(h_console, &console_info);
11.    saved_attributes = console_info.wAttributes;
12.
13.    menuDrawer = new MenuDrawer(this);
14.    helpDrawer = new HelpDrawer(this);
15.    fileCreator = new FileCreator(this);
16.    fileRenamer = new FileRenamer(this);
17.    fileDeleter = new FileDeleter(this);
18.    filePreviewer = new FilePreviewer(this);
19.    fileMover = new FileMover(this);
20.
21.    SetConsoleCP(RUSSIAN_CP);
22.    SetConsoleOutputCP(RUSSIAN_CP);
23.    utils.clearScreen();
24.
25.    list_files = fd.GetFilesList();
26.    redrawConsoleGui();
27. }
28.
29. ConsoleGuiHandler::~ConsoleGuiHandler() {
30.     SetConsoleTextAttribute(h_console, saved_attributes);
31.     std::cin.clear();
32.     fflush(stdin);
33.     utils.clearScreen();
34.     CloseHandle(h_console);
35. }
36.

```



```

37. void ConsoleGuiHandler::redrawConsoleGui() {
38.     menuDrawer->redrawConsoleGui();
39. }
40.
41. void ConsoleGuiHandler::cleanRedrawConsoleGui() {
42.     menuDrawer->cleanRedrawConsoleGui();
43. }
44.
45. void ConsoleGuiHandler::moveDown() {
46.     int relative_index = current_selected_index + 1 - starting_index;
47.     if (relative_index < menuDrawer-
>MAX_FILES && current_selected_index < list_files.size() - 1) {
48.         current_selected_index++;
49.     } else if (relative_index == menuDrawer-
>MAX_FILES && current_selected_index < list_files.size() - 1) {
50.         current_selected_index++;
51.         starting_index += menuDrawer->MAX_FILES;
52.     } else if (current_selected_index == list_files.size() - 1) {
53.         starting_index = 0;
54.         current_selected_index = 0;
55.     }
56.     redrawConsoleGui();
57. }
58.
59. void ConsoleGuiHandler::moveUp() {
60.     int relative_index = current_selected_index + 1 - starting_index;
61.     if (relative_index > 1) {
62.         current_selected_index--;
63.     }
64.     else if (relative_index == 1 && starting_index > 0) {
65.         current_selected_index--;
66.         starting_index -= menuDrawer->MAX_FILES;
67.     }
68.     else if (relative_index == 1 && starting_index == 0) {
69.         current_selected_index = list_files.size() - 1;
70.         starting_index = current_selected_index / menuDrawer->MAX_FILES * menuDrawer-
>MAX_FILES;
71.     }
72.     redrawConsoleGui();
73. }
74.
75. void ConsoleGuiHandler::goUp() {
76.     reInitSafe("..");
77. }
78.
79. void ConsoleGuiHandler::open() {
80.     std::string file_name = list_files[current_selected_index].getName();
81.     switch(list_files[current_selected_index].getType()) {
82.         case FileType::DIR:
83.             reInitSafe(file_name);
84.             break;
85.         case FileType::EXE:
86.         case FileType::ORD:
87.             std::string command = R"(start "" ")" + file_name + "\"";
88.             system(command.c_str());
89.             break;
90.     }
91. }
92.
93. void ConsoleGuiHandler::openDir() {
94.     std::string file_name = list_files[current_selected_index].getName();
95.     if(list_files[current_selected_index].getType() == FileType::DIR)
96.         reInitSafe(file_name);
97. }
98.
99. void ConsoleGuiHandler::reInit(std::string file_name) {
100.     redrawConsoleGui();

```

```

101.     //file_name = fd.getCurrentDirectory() + "/" + file_name;
102.     FiledirectoryException e = fd.reInit(file_name);
103.     if(e == FiledirectoryException::NO_EXCEPTION) {
104.         list_files = fd.GetFilesList();
105.         if(current_selected_index >= list_files.size()) {
106.             current_selected_index = list_files.size() - 1;
107.         }
108.         if(starting_index >= list_files.size() && list_files.size() != 0) {
109.             starting_index -= menuDrawer->MAX_FILES;
110.         }
111.         redrawConsoleGui();
112.     } else {
113.         outputCorrespondingException(e);
114.     }
115. }
116.
117. void ConsoleGuiHandler::reInitSafe(std::string file_name) {
118.     redrawConsoleGui();
119.     //file_name = fd.getCurrentDirectory() + "/" + file_name;
120.     FiledirectoryException e = fd.reInit(file_name);
121.     if(e == FiledirectoryException::NO_EXCEPTION) {
122.         list_files = fd.GetFilesList();
123.         current_selected_index = 0;
124.         starting_index = 0;
125.         redrawConsoleGui();
126.     } else {
127.         outputCorrespondingException(e);
128.     }
129. }
130.
131. void ConsoleGuiHandler::outputCorrespondingException(FiledirectoryException e) {
132.     redrawConsoleGui();
133.     switch(e) {
134.         case FiledirectoryException::ACCESS_DENIED:
135.             utils.outputLine("Access is denied!", saved_attributes);
136.             break;
137.         case FiledirectoryException::FILE_NOT_FOUND:
138.             utils.outputLine("File not found!", saved_attributes);
139.             break;
140.         case FiledirectoryException::INCORRECT_NAME:
141.             utils.outputLine("Preserved name!", saved_attributes);
142.             break;
143.         case FiledirectoryException::UNHANDLED:
144.             utils.outputLine("An error has occurred!", saved_attributes);
145.             break;
146.     }
147. }
148.
149. bool ConsoleGuiHandler::checkFile(const std::string &name) {
150.     if(name.empty()) return false;
151.     if(fd.containsCurrent(name)) {
152.         redrawConsoleGui();
153.         utils.outputLine(FD_EXISTS, saved_attributes);
154.         return false;
155.     }
156.     return true;
157. }
158.
159. void ConsoleGuiHandler::rename() {
160.     fileRenamer->rename();
161. }
162.
163. void ConsoleGuiHandler::createFileOrDir() {
164.     fileCreator->createFileOrDir();
165. }
166.
167. void ConsoleGuiHandler::deleteFile() {

```

```

168.     fileDeleter->deleteFile();
169. }
170.
171. void ConsoleGuiHandler::showHelp() {
172.     helpDrawer->showHelp();
173. }
174.
175. void ConsoleGuiHandler::showTextPreview() {
176.     filePreviewer->showTextPreview();
177. }
178.
179. void ConsoleGuiHandler::showRawPreview() {
180.     filePreviewer->showRawPreview();
181. }
182.
183. void ConsoleGuiHandler::saveLocation() {
184.     fileMover->saveLocation();
185. }
186.
187. void ConsoleGuiHandler::moveFile() {
188.     fileMover->moveFile();
189. }
190.
191. void ConsoleGuiHandler::copyFile() {
192.     fileMover->copyFile();
193. }

```

## Класс ConsoleGuiUtils

```

1. #include "../headers/gui/ConsoleGuiUtils.h"
2. #include <conio.h>
3.
4. ConsoleGuiUtils::ConsoleGuiUtils(HANDLE h_console) : h_console(h_console) {}
5.
6. void ConsoleGuiUtils::clearScreen() {
7.     COORD tl = { 0,0 };
8.     CONSOLE_SCREEN_BUFFER_INFO s;
9.     GetConsoleScreenBufferInfo(h_console, &s);
10.    DWORD written, cells = s.dwSize.X * s.dwSize.Y;
11.    FillConsoleOutputCharacter(h_console, ' ', cells, tl, &written);
12.    FillConsoleOutputAttribute(h_console, s.wAttributes, cells, tl, &written);
13.    SetConsoleCursorPosition(h_console, tl);
14. }
15.
16. void ConsoleGuiUtils::clearCurrentLine() {
17.     CONSOLE_SCREEN_BUFFER_INFO s;
18.     GetConsoleScreenBufferInfo(h_console, &s);
19.     COORD current_line_coord = { 0, s.dwCursorPosition.Y };
20.     clearLine(current_line_coord);
21. }
22.
23. void ConsoleGuiUtils::clearLine(COORD current_line_coord) {
24.     CONSOLE_SCREEN_BUFFER_INFO s;
25.     GetConsoleScreenBufferInfo(h_console, &s);
26.     DWORD written, cells = s.dwSize.X;
27.     FillConsoleOutputCharacter(h_console, ' ', cells, current_line_coord, &written);
28.     FillConsoleOutputAttribute(h_console, 0, cells, current_line_coord, &written);
29. }
30.
31. void ConsoleGuiUtils::outputLine(ConsoleLine line) {
32.     CONSOLE_SCREEN_BUFFER_INFO s;
33.     GetConsoleScreenBufferInfo(h_console, &s);
34.     COORD current_cursor_coord = s.dwCursorPosition;
35.     COORD next_line_coord = { 0, (short)(s.dwCursorPosition.Y + 1) };
36.     DWORD written;

```

```

37.     WriteConsoleA(h_console, line.getText().c_str(), line.getText().size(), &written, null
ptr);
38.     FillConsoleOutputAttribute(h_console, line.getAttribute(), line.getText().size(), curr
ent_cursor_coord, &written);
39.     SetConsoleCursorPosition(h_console, next_line_coord);
40. }
41.
42. void ConsoleGuiUtils::outputLine(const std::string& str, WORD attr) {
43.     outputLine(ConsoleLine(str, attr));
44. }
45.
46. void ConsoleGuiUtils::outputLineNoNew(ConsoleLine line) {
47.     CONSOLE_SCREEN_BUFFER_INFO s;
48.     GetConsoleScreenBufferInfo(h_console, &s);
49.     COORD current_cursor_coord = s.dwCursorPosition;
50.     DWORD written;
51.     WriteConsoleA(h_console, line.getText().c_str(), line.getText().size(), &written, null
ptr);
52.     FillConsoleOutputAttribute(h_console, line.getAttribute(), line.getText().size(), curr
ent_cursor_coord, &written);
53. }
54.
55. void ConsoleGuiUtils::outputLineNoNew(const std::string& str, WORD attr) {
56.     outputLineNoNew(ConsoleLine(str, attr));
57. }
58.
59. void ConsoleGuiUtils::outputChar(const char c, WORD attr) {
60.     CONSOLE_SCREEN_BUFFER_INFO s;
61.     GetConsoleScreenBufferInfo(h_console, &s);
62.     COORD current_cursor_coord = s.dwCursorPosition;
63.     DWORD written;
64.     WriteConsole(h_console, &c, 1, &written, nullptr);
65.     FillConsoleOutputAttribute(h_console, attr, 1, current_cursor_coord, &written);
66. }
67.
68. std::string ConsoleGuiUtils::inputLine(WORD attr) {
69.     char buff[INPUT_MAX_LENGTH] = {0};
70.     int index = 0;
71.     int c;
72. iloop:
73.     c = getch();
74.     switch(c) {
75.         case 27: //esc
76.             return "";
77.         case 0: //extended
78.             getch();
79.             goto iloop;
80.         case '\n':
81.         case '\r':
82.             return buff;
83.         case '\b':
84.             outputChar(c, attr);
85.             outputChar(' ', attr);
86.             outputChar(c, attr);
87.             index -= index != 0;
88.             buff[index] = 0;
89.             goto iloop;
90.         default:
91.             if(index < INPUT_MAX_LENGTH) {
92.                 outputChar(c, attr);
93.                 buff[index] = c;
94.                 index++;
95.             }
96.             goto iloop;
97.     }
98. }

```

## Класс ConsoleLine

```
1. #include "../headers/gui/ConsoleLine.h"
2.
3. #include <utility>
4.
5. ConsoleLine::ConsoleLine() : attribute(0) { }
6.
7. ConsoleLine::ConsoleLine(std::string text, WORD attribute) : text(std::move(text)), attribute(attribute) { }
8.
9. void ConsoleLine::setText(std::string _text) {
10.     text = std::move(_text);
11. }
12.
13. void ConsoleLine::setAttribute(WORD _attribute) {
14.     attribute = _attribute;
15. }
16.
17. std::string ConsoleLine::getText() {
18.     return text;
19. }
20.
21. WORD ConsoleLine::getAttribute() const {
22.     return attribute;
23. }
24.
25. bool ConsoleLine::operator==(const ConsoleLine &other) {
26.     if(this->text == other.text && this->attribute == other.attribute)
27.         return true;
28.     return false;
29. }
30.
31. bool ConsoleLine::operator!=(const ConsoleLine &other) {
32.     return !(*this == other);
33. }
```

## Класс KeypressHandler

```
1. #include "../headers/KeypressHandler.h"
2. #include <conio.h>
3.
4. KeypressHandler::KeypressHandler(ConsoleGuiHandler *cgh) : cgh(cgh) {
5. }
6.
7. int KeypressHandler::start() {
8.     int c;
9.     while(kbhit()) getch();
10.
11. mloop:
12.     c = getch();
13.     switch(c) {
14.         case 0: //func keys
15.         case 224:
16.             c = getch();
17.             switch(c) {
18.                 case 72: //up arrow
19.                     cgh->moveUp();
20.                     break;
21.                 case 80: //down arrow
22.                     cgh->moveDown();
23.                     break;
24.                 case 75: //left arrow
25.                     cgh->goUp();
```

```

26.         break;
27.     case 77: //right arrow
28.         cgh->openDir();
29.         break;
30.     case 60: //f2
31.         cgh->showHelp();
32.         break;
33.     }
34.     goto mloop;
35. case 'h':
36.     cgh->goUp();
37.     goto mloop;
38. case 'j':
39.     cgh->moveDown();
40.     goto mloop;
41. case 'k':
42.     cgh->moveUp();
43.     goto mloop;
44. case 'l':
45.     cgh->openDir();
46.     goto mloop;
47. case 'v':
48.     cgh->showTextPreview();
49.     goto mloop;
50. case 'V':
51.     cgh->showRawPreview();
52.     goto mloop;
53. case 'q':
54. case 'Q':
55.     return 0;
56. case 'o':
57. case 'O':
58.     cgh->open();
59.     goto mloop;
60. case 'r':
61. case 'R':
62.     cgh->rename();
63.     goto mloop;
64. case 'n':
65. case 'N':
66.     cgh->createFileOrDir();
67.     goto mloop;
68. case 'd':
69. case 'D':
70.     cgh->deleteFile();
71.     goto mloop;
72. case 'c':
73. case 'C':
74.     cgh->saveLocation();
75.     goto mloop;
76. case 'm':
77. case 'M':
78.     cgh->moveFile();
79.     goto mloop;
80. case 'p':
81. case 'P':
82.     cgh->copyFile();
83.     goto mloop;
84. default:
85.     goto mloop;
86. }
87. }

```

## **ПРИЛОЖЕНИЕ Б**