

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

**БИЗНЕС-ПРОЕКТ «ПРОГРАММА ДЛЯ
ПОБИТОВОГО СРАВНЕНИЯ МНОЖЕСТВА
ФАЙЛОВ БОЛЬШОГО ОБЪЕМА»**

Студент Кожняков Дмитрий Михайлович
(подпись, дата) (ФИО полностью)

Руководитель Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 16)

Заведующий кафедрой М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от
09.06.2025 г. № 21)

Секретарь ГЭК Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

М.А. Митрохин

«06» мая 2025 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Кожняков Дмитрий Михайлович гр. 21ВВП1 факультета ФВТ

2. Тема работы бизнес-проект «программа для побитового сравнения множества файлов большого объема»

Тема утверждена приказом ПГУ № 601/о от " 06 " мая 2025 г.

3. Руководитель бакалаврской работы Бычков А.С.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать программное обеспечение для побитового сравнения файлов большого объема. Составить бизнес-модель и финансовый план. Язык программирования C++, среда разработки Visual Studio 2019. Программа должна обладать интуитивно понятным интерфейсом и использовать следующие технологии:

1. Windows API

2. C/C++

3. Асинхронное программирование

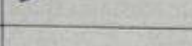
4. Реестр Windows

5. Многопоточность

5. Перечень подлежащих разработке вопросов

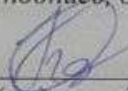
Требуется произвести анализ предметных областей: изучить существующие информационные системы, выделить их достоинства и недостатки. Выявить потребности пользователей. Выбрать язык программирования, среду разработки. Определить архитектуру приложения и вспомогательные средства разработки. Изучить функционал библиотек для разработки приложения на C/C++.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	4%	06.05.2025	
2	Анализ требований	4%	07.05.2025	
3	Выбор технологий, применяемых при проектировании и разработке	2%	08.05.2025	
4	Проектирование архитектуры	10%	09.05.2025 - 11.05.2025	
5	Проектирование и Разработка ПО	50%	12.05.2025 - 24.05.2025	
6	Тестирование и отладка	20%	25.05.2025 - 29.05.2025	
7	Оформление выпускной работы	10%	30.05.2025	

Дата выдачи задания "06" мая 2025 г.

Руководитель бакалаврской работы  Бычков А.С.
(подпись, дата)

Задание к исполнению принял студент  Кожняков Д.М.
(подпись, дата)

Работу к защите допустить
Декан факультета  06.06.2025 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 112 листа, 34 рисунка, 1 таблицу, 5 источников, 3 приложения.

ПОБИТОВОЕ СРАВНЕНИЕ, СРАВНЕНИЕ ФАЙЛОВ, БОЛЬШИЕ ФАЙЛЫ, МНОЖЕСТВЕННОЕ СРАВНЕНИЕ, C++, WINAPI, VISUAL STUDIO, MEMORY-MAPPED FILES, СКОЛЬЗЯЩЕЕ ОКНО, ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС, ВИЗУАЛИЗАЦИЯ ДАННЫХ, УПРАВЛЕНИЕ ПАМЯТЬЮ, РЕЕСТР WINDOWS, МНОГОПОТОЧНОСТЬ

Цель работы – разработка специализированного программного обеспечения, предназначенного для высокоточного побитового сравнения файлов значительного объема. Достижение поставленной цели предполагает создание эффективного десктопного приложения, использующего возможности низкоуровневого программирования на языке C/C++ в сочетании с функционалом Windows API, что позволит обеспечить надежность, скорость анализа данных и удобство взаимодействия пользователя.

Разрабатываемая программа призвана предоставить пользователю мощный инструмент для верификации и анализа целостности данных путем побайтового сравнения множества файлов. Особое внимание уделяется оптимизации производительности при работе с файлами большого объема и реализации интуитивно понятного интерфейса для визуализации обнаруженных различий.

В результате выполнения выпускной квалификационной работы будет разработано функциональное и производительное десктопное приложение, способное эффективно решать задачи сравнения файлов. Программный продукт, реализованный с применением языков C/C++ и низкоуровневых функций Windows API, будет полностью соответствовать поставленной цели, а его тестирование подтвердит готовность к практическому использованию.

					ПГУ1.090301.10.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Бизнес-проект «программа для побитового сравнения множества файлов большого объема» Пояснительная записка	Лит.	Лист	Листов
Разраб.		Кожняков Д.М.		14.06.25				
Провер.		Бычков А.С.		14.06.25			2	112
Н. Контр.		Бычков А.С.		14.06.25		ФВТ, каф. ВТ гр. 21ВВП1		
Утверд.		Митрохин М.А.		14.06.25				

Содержание

Перечень сокращений и обозначений.....	4
Введение.....	6
1 Бизнес-проект «программа для побитового сравнения множества файлов большого объема»	8
1.1 Анализ предметной области	8
1.2 Анализ решения поставленной задачи	10
1.3 Анализ актуальности разработки	11
1.4 Анализ аналогов информационных систем	13
1.5 Анализ архитектуры и структуры программы.....	17
1.6 Постановка задачи	20
1.7 Выбор инструментов разработки	21
1.8 Реализация	24
1.8.1. Создание проекта	24
1.8.2. Описание доступных функций	27
1.8.3. Описание программы.....	39
2 Безопасность жизнедеятельности.....	43
2.1 Правила безопасности при использовании ПК.....	43
3 Стартап проект	45
3.1 Сегменты ЦА	45
3.2 Ключевые процессы.....	46
3.3 Каналы взаимодействия	48
3.4 Источники дохода	50
3.5 Ключевые ресурсы.....	51
3.6 Риски проекта	52
Заключение	54
Список использованных источников	56
Приложение А – UML диаграммы	57
Приложение Б – Листинг программы	60
Приложение В – Презентация.....	108

Перечень сокращений и обозначений

В настоящей выпускной квалификационной работе применяют следующие сокращения и обозначения

API (Application Programming Interface) – описание способов, которыми одна компьютерная программа может взаимодействовать с другой программой.

Big Data (Большие данные) – совокупность подходов, инструментов и методов обработки структурированных и неструктурированных данных огромных объемов и значительного многообразия для получения результатов, воспринимаемых как ценность

Diff (от англ. difference) – утилита или алгоритм сравнения, предназначенный для выявления различий между двумя или более файлами, обычно текстовыми, с отображением этих различий построчно

IDE (Integrated Development Environment) – интегрированная среда разработки.

Memory-mapped file (отображаемый в память файл) – метод файлового ввода/вывода, позволяющий сопоставить (отобразить) содержимое файла напрямую с областью виртуальной памяти процесса, что позволяет работать с файлом как с массивом в оперативной памяти

Скролл (прокрутка) — это механизм взаимодействия с пользовательским интерфейсом, позволяющий перемещаться по содержимому, которое превышает размеры видимой области (окна, панели, элемента управления). Он реализуется путем смещения отображаемого контента вверх/вниз (вертикальная прокрутка) или влево/вправо (горизонтальная прокрутка) для доступа к невидимым частям информации

ОЗУ – оперативное запоминающее устройство

ПК – персональный компьютер

ПО – программное обеспечение

ЦП – центральный процессор

ЭВМ – электронно-вычислительная машина

					ПГУ1.090301.10.001 ПЗ	Лист
						4
Изм.	Лист	№ докум.	Подпись	Дата		

Скользящее окно (Sliding Window) – техника обработки данных, при которой данные обрабатываются последовательными, перекрывающимися или смежными блоками (окнами) фиксированного размера, перемещающимися по всему объему данных

WinAPI (Windows Application Programming Interface) – набор функций операционной системы Microsoft Windows, предоставляющий приложениям доступ к её ресурсам и службам

					ПГУ1.090301.10.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

В современном цифровом мире, где объемы генерируемых и хранимых данных неуклонно растут, вопросы целостности, аутентичности и верификации информации приобретают первостепенное значение. От надежности хранения и точности передачи данных зависит функционирование критически важных систем, будь то финансовые транзакции, научные исследования, программное обеспечение или архивные фонды. В условиях динамичного развития информационных технологий и повсеместного распространения «больших данных» (Big Data), потребность в инструментах, способных эффективно и точно сравнивать файлы, становится все более острой.

Традиционные методы сравнения, основанные на контрольных суммах (хешах) или сравнении по строкам (для текстовых файлов), не всегда обеспечивают необходимый уровень детализации и достоверности. Хеш-суммы лишь подтверждают полную идентичность или различия, не указывая на конкретное место и характер изменений. Сравнение текстовых файлов не применимо к бинарным данным, которые составляют значительную часть цифровой информации (исполняемые файлы, архивы, образы дисков, мультимедиа). Более того, при работе с файлами большого объема (гигабайты и терабайты) эти методы часто оказываются неэффективными с точки зрения производительности и потребления системных ресурсов.

Цель работы – разработка специализированного программного обеспечения, предназначенного для высокоточного побитового сравнения файлов значительного объема. Достижение поставленной цели предполагает создание эффективного десктопного приложения, использующего возможности низкоуровневого программирования на языке C/C++ в сочетании с функционалом Windows API, что позволит обеспечить надежность, скорость анализа данных и удобство взаимодействия пользователя.

Для достижения поставленной цели в рамках выпускной квалификационной работы были определены следующие задачи:

					ПГУ1.090301.10.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

- освоение современных средств и технологий программной разработки, применимых для создания высокопроизводительных десктопных приложений;
- проведение анализа предметной области, выявление ключевых проблем и требований, предъявляемых к программам побитового сравнения файлов большого объема;
- исследование существующих программных аналогов, выявление их преимуществ и недостатков с целью определения оптимальных подходов к реализации и выработки уникальных особенностей разрабатываемого решения;
- формирование требований к функциональности и пользовательскому интерфейсу разрабатываемой программы, учитывая специфику работы с бинарными данными и потребность в сравнении множества файлов;
- разработка архитектуры программного продукта, включающей модули для эффективного чтения, побитового сравнения и визуализации данных;
- реализация прототипа программы для побитового сравнения файлов большого объема с использованием C/C++ и Windows API, включая механизм memory-mapped files;
- проведение тестирования разработанного программного обеспечения для подтверждения его функциональности, производительности и надежности при работе с различными объемами и типами файлов;
- анализ полученных результатов, формулирование выводов о достижении цели работы и определение перспектив дальнейшего развития программного продукта.

1 Бизнес-проект «Программа для побитового сравнения множества файлов большого объема»

1.1 Анализ предметной области

Для разработки высокоэффективного и функционального программного обеспечения для побитового сравнения файлов необходимо провести глубокий анализ предметной области, выявить ключевые потребности пользователей и определить требования к системе. В рамках данного анализа была произведена оценка существующих методов работы с файловыми данными, выявлены ограничения традиционных подходов и сформулированы задачи, которые могут быть оптимизированы с помощью специализированного программного продукта.

Анализ предметной области включает в себя рассмотрение основных процессов, связанных с управлением файлами, где критически важна их целостность и возможность детального сравнения. К таким процессам относятся, например, контроль версий программного обеспечения, где разработчики нуждаются в точной идентификации изменений между различными версиями исходного кода или исполняемых файлов; верификация резервных копий и архивов, требующая гарантии идентичности скопированных или заархивированных данных с оригиналами; обнаружение повреждений данных или вредоносных изменений, что важно для специалистов по информационной безопасности; а также синхронизация данных в распределенных системах и цифровая криминалистика, где требуется абсолютная точность при анализе цифровых доказательств.

Далее были определены потребности и требования различных категорий пользователей к программе побитового сравнения. В частности, к ним относятся необходимость высокой точности выявления любых отличий до уровня бита, эффективная работа с файлами большого объема (гигабайты и терабайты) без значительного потребления системных ресурсов, поддержка одновременного сравнения нескольких файлов для экономии времени и упрощения анализа, а

					ПГУ1.090301.10.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

также наглядная визуализация различий с использованием цветового кодирования и отображением данных в различных представлениях (HEX и символьном). Не менее важными являются удобная навигация по найденным отличиям, универсальность, позволяющая сравнивать файлы любого типа, и общая высокая производительность программы.

В ходе анализа были выявлены основные проблемы и недостатки существующих подходов и программных решений, которые могут быть эффективно решены с помощью разрабатываемой системы. Эти проблемы включают ограничения по объему обрабатываемых файлов, что делает многие утилиты непригодными для «больших данных»; отсутствие специализированных функций побитового сравнения для бинарных файлов в большинстве стандартных diff-утилит; недостаточную детализацию результатов, когда программы на основе хеш-сумм не указывают место изменений; ограничения по количеству одновременно сравниваемых файлов; низкую производительность неоптимизированных алгоритмов; а также неудобный или устаревший пользовательский интерфейс, затрудняющий интерпретацию результатов.

На основе проведенного анализа были сформулированы следующие требования к новой системе. Программа должна обеспечивать побитовое сравнение до 9 файлов одновременно. Необходимо реализовать эффективные алгоритмы чтения файлов большого объема, использующие механизм memory-mapped files с поблочной обработкой данных. Важным аспектом является предоставление интуитивно понятного графического интерфейса пользователя с функциями навигации по отличиям («вперед/назад») и глобальной оценки идентичности. Требуется внедрение гибкой системы визуализации различий с использованием цветового кодирования для выделения уникальных и совпадающих участков данных.

В результате анализа предметной области стало очевидно, что разработка специализированной программы для побитового сравнения файлов большого объема с учетом указанных требований является актуальной и востребованной

					ПГУ1.090301.10.001 ПЗ	Лист
						9
Изм.	Лист	№ докум.	Подпись	Дата		

задачей, способной существенно оптимизировать процессы контроля и анализа файловых данных.

1.2 Анализ решения поставленной задачи

Для разработки высокопроизводительной и эффективной программы побитового сравнения файлов большого объема был сделан выбор в пользу языка программирования C/C++ и средств Windows API. Данная комбинация технологий является оптимальной для решения поставленной задачи, поскольку предоставляет уникальные возможности для низкоуровневой работы с данными и ресурсами операционной системы.

C/C++ – это мощные языки программирования, которые обеспечивают беспрецедентный контроль над системными ресурсами и памятью. Их использование является основополагающим для создания инструмента, способного обрабатывать файлы объемом в гигабайты и терабайты с максимальной скоростью. Ключевые преимущества C/C++ в контексте данной разработки включают:

- высокая производительность: компиляция в машинный код позволяет программе выполнять побитовые операции и операции файлового ввода-вывода с минимальными задержками, что критично для обработки больших объемов данных;

- эффективное управление памятью: возможность непосредственного управления выделением и освобождением памяти дает разработчику полный контроль над потреблением ОЗУ, предотвращая избыточное использование ресурсов, что особенно важно при работе со «скользящим окном» memory-mapped files;

- прямой доступ к аппаратным ресурсам: языки предоставляют средства для прямого взаимодействия с файловой системой и другими компонентами операционной системы на низком уровне, что необходимо для оптимизации производительности.

Windows API (Application Programming Interface) – это набор функций, предоставляемых операционной системой Microsoft Windows, которые позволяют прикладным программам взаимодействовать с её базовыми службами и функциями. Применение WinAPI в разрабатываемой программе обусловлено следующими преимуществами:

- низкоуровневые операции ввода-вывода: WinAPI предоставляет доступ к функциям асинхронного ввода-вывода (ReadFileEx, WriteFileEx и другие), которые позволяют выполнять операции чтения и записи данных в фоновом режиме, не блокируя основной поток приложения. Это значительно повышает отзывчивость интерфейса и общую скорость работы программы при сравнении нескольких файлов;

- механизм Memory-mapped files: Одна из наиболее мощных функций WinAPI для работы с большими файлами, позволяющая отображать содержимое файла напрямую в виртуальное адресное пространство процесса. Это устраняет необходимость в традиционных буферизированных операциях чтения и записи, позволяя программе работать с файлом как с массивом в памяти, что значительно ускоряет доступ к данным и упрощает логику их обработки.

Интеграция с ОС: использование WinAPI обеспечивает тесную интеграцию программы с операционной системой Windows, гарантируя стабильность, совместимость и возможность использования всех системных оптимизаций.

1.3 Анализ актуальности разработки

Актуальность разработки программы для побитового сравнения файлов большого объема определяется рядом факторов, обусловленных современными тенденциями в сфере информационных технологий и управления данными.

Во-первых, наблюдается экспоненциальный рост объемов цифровой информации, хранимой и передаваемой в различных форматах. Это касается как структурированных, так и неструктурированных данных, включая программный код, мультимедийные файлы, образы дисков, базы данных и архивные хранилища.

Увеличение объемов данных неизбежно порождает потребность в надежных и эффективных инструментах для контроля их целостности, верификации и обнаружения даже мельчайших изменений.

Во-вторых, традиционные подходы к сравнению файлов, такие как использование контрольных сумм или построчное сравнение для текстовых данных, имеют существенные ограничения. Хеш-суммы позволяют лишь констатировать факт идентичности или различия файлов, не предоставляя информации о конкретном местоположении и характере изменений. При этом они не позволяют эффективно сравнивать различные версии одного файла, показывая лишь, что файлы «разные». Построчное сравнение, хоть и более детализировано, неприменимо к бинарным файлам, которые составляют значительную часть современного информационного потока.

В-третьих, современные задачи часто требуют не только сравнения двух файлов, но и анализа различий между множеством версий или копий одного и того же файла. Существующие аналогичные программные продукты зачастую ограничены функционалом парного сравнения, не справляясь с одновременным анализом трех и более файлов, что значительно снижает их эффективность в сценариях, требующих массовой проверки данных или работы с сложными версионированными системами.

Наконец, производительность и удобство использования остаются критически важными аспектами. Неоптимизированные решения не способны эффективно работать с файлами, объемы которых могут исчисляться в сотни гигабайт или терабайт, приводя к длительным задержкам и избыточному потреблению системных ресурсов. Кроме того, интерфейс многих существующих программ является либо устаревшим, либо недостаточно информативным для наглядного отображения побитовых различий.

Таким образом, разработка специализированной программы, способной осуществлять побитовое сравнение множества файлов большого объема с высокой производительностью, оптимизированным потреблением ресурсов.

1.4 Анализ аналогов информационных систем

Для формирования полноценного представления о текущем состоянии рынка программного обеспечения для сравнения файлов и определения уникальных преимуществ разрабатываемой системы был проведен анализ существующих аналогичных решений. Были рассмотрены как специализированные утилиты, так и интегрированные инструменты, предоставляющие функционал сравнения данных.

Total Commander – это популярный файловый менеджер для Windows, который в числе прочего функционала включает встроенную утилиту для сравнения файлов. Инструмент сравнения интегрирован в удобный и широко используемый файловый менеджер, что обеспечивает быстрый доступ к функции без необходимости запуска отдельного приложения. Пользовательский интерфейс сравнения достаточно интуитивен для базовых задач. Однако, Total Commander позволяет сравнивать только два файла одновременно, что значительно ограничивает его применимость в сценариях, требующих анализа множества версий или копий данных. Также существует ограничение на размер сравниваемых файлов (как правило, до 150 МБ), что делает его непригодным для работы с большими объемами данных (гигабайты и терабайты). Сравнение файлов является лишь дополнительной, а не основной функцией Total Commander, что сказывается на глубине проработки алгоритмов и специфических возможностях для побитового анализа. Кроме того, отсутствует детальная побитовая визуализация, поскольку инструмент ориентирован на текстовое сравнение или базовую проверку бинарных файлов без предоставления детального побитового представления отличий (например, в HEX-формате с указанием смещения). Пример работы аналога «Total Commander» рассмотрен на рисунке 1.1.

рутинных задач анализа различий, требуя от пользователя значительных усилий и времени для выявления несоответствий. Программа предназначена скорее для экспертов, работающих с данными на очень низком уровне, а не для рядового пользователя, которому требуется быстро и наглядно увидеть отличия. Пример работы аналога «WinHex» рассмотрен на рисунке 1.2.

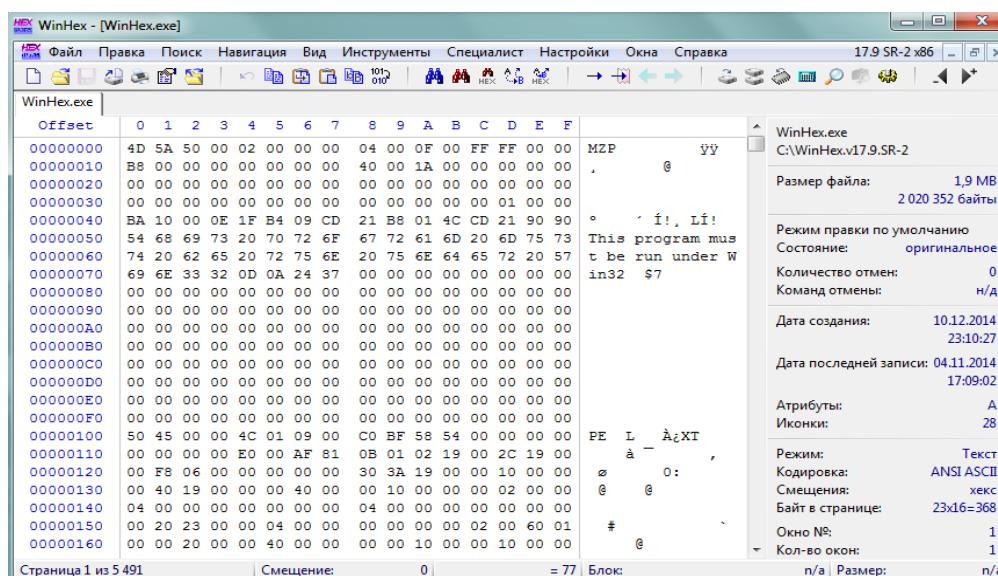


Рисунок 1.2 – Обзор аналога «WinHex»

DiffMerge – это специализированная утилита для визуального сравнения и объединения файлов и папок, часто используемая разработчиками. DiffMerge ориентирован на подробное и наглядное отображение различий в текстовых файлах, предоставляя возможность редактирования текста непосредственно в процессе сравнения. Его основной задачей является визуализация изменений и помощь в слиянии версий. Однако, главным недостатком для нашей задачи является отсутствие возможности просмотра и сравнения файлов в шестнадцатеричном виде, что делает его непригодным для побитового сравнения бинарных файлов. Как и Total Commander, DiffMerge обычно позволяет сравнивать только два файла, что не соответствует требованию по сравнению множества файлов. Пользовательский интерфейс программы также может показаться устаревшим, что влияет на удобство работы и общую эстетику. Основной фокус программы на текстовых файлах ограничивает ее

универсальность для побитового анализа любых данных. Пример работы аналога «DiffMerge» рассмотрен на рисунке 1.3.

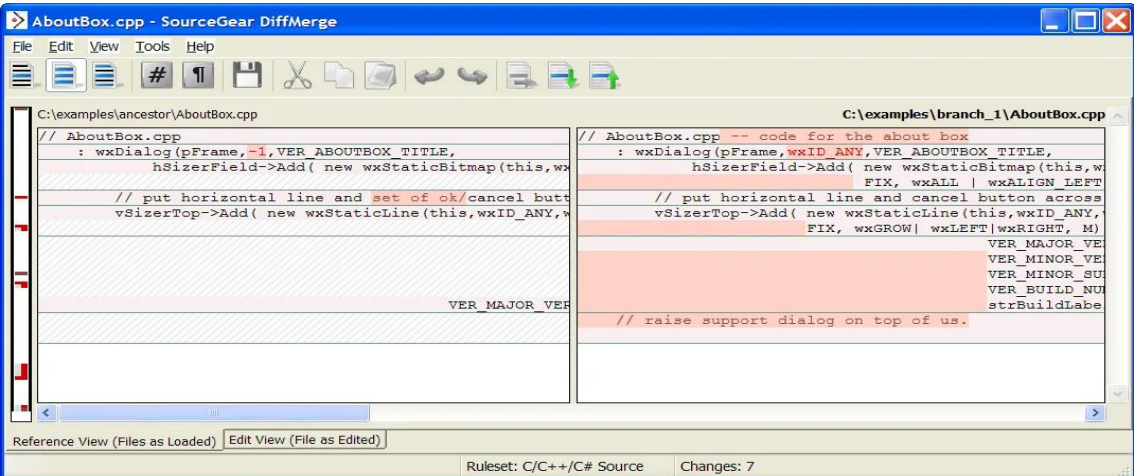


Рисунок 1.3 – Обзор аналога «DiffMerge»

Проведенный анализ существующих решений показал, что ни один из рассмотренных аналогов не обладает полным набором функциональных возможностей, необходимых для эффективного и универсального побитового сравнения множества файлов большого объема. Основные недостатки включают: ограничения по количеству сравниваемых файлов, неспособность работать с большими объемами данных, отсутствие или недостаточную детализацию побитового отображения, а также ограниченную универсальность для бинарных файлов. Разрабатываемая программа призвана преодолеть эти недостатки, предоставив пользователю инструмент, который способен сравнивать более 2 файлов одновременно, эффективно работает с файлами любого, обеспечивает точное побитовое сравнение и наглядную визуализацию отличий в HEX и символьном представлениях с помощью продуманной цветовой схемы, а также обладает высокой производительностью. В таблице 1 можно увидеть краткое содержание плюсов и минусов программ аналогов.

Таблица 1 – Краткое содержание аналоговых решений

Название аналога	Преимущества	Недостатки
Total commander	+удобный интерфейс +подробное отображение отличий	-можно сравнивать только 2 файла -размер файла до 150мб
WinHex	+отображение того, чем является данный кусок данных (например имя файла) +может открывать файлы и диски любых размеров	-нет возможности сравнивать файлы
DiffMerge	+возможность редактирования текста, во время сравнения +подробное отображение отличий	-отсутствие 16-ого представление файлов -можно сравнивать только 2 файла -устаревший интерфейс

1.5 Анализ архитектуры и структуры программы

Разработка эффективного программного обеспечения для побитового сравнения файлов большого объема требует тщательно продуманной архитектуры, обеспечивающей как высокую производительность, так и удобство взаимодействия с пользователем. В рамках данной работы было принято решение о создании десктопного приложения, построенного на монолитной архитектуре.

Монолитная архитектура была выбрана ввиду её преимуществ для проекта такого масштаба и специализации. Она характеризуется тем, что все компоненты приложения (пользовательский интерфейс, бизнес-логика, модуль обработки данных) объединены в единый исполняемый файл. Это обеспечивает

автономность и независимость приложения, исключает необходимость установки дополнительных сервисов или зависимостей. Ключевыми достоинствами монолитной архитектуры для данной программы являются: высокая производительность за счет минимизации межпроцессных взаимодействий, простота отладки и тестирования всего функционала как единого целого, а также удобство развертывания, поскольку приложение не требует сложных настроек инфраструктуры.

Структура программы включает в себя несколько ключевых модулей, работающих в тесном взаимодействии:

Модуль пользовательского интерфейса (UI): Этот модуль отвечает за взаимодействие программы с пользователем. Он реализуется в виде графического окна, центральной частью которого являются области для отображения содержимого сравниваемых файлов. В качестве элементов управления предусмотрены кнопки для открытия файлов («Открыть файл 1», «Открыть файл 2» и т.д.), запуска процесса сравнения, навигации по обнаруженным отличиям («Найти отличие вперед», «Найти отличие назад»). Также предусмотрено «Меню настроек», позволяющее пользователю адаптировать программу под свои нужды, включая выбор количества сравниваемых файлов, настройку шрифтов для отображения данных и изменение цветовой темы интерфейса для улучшения эргономики. Целью UI является максимальное удобство и эффективность работы с побитовым сравнением, а также простота и понятность даже для пользователей без глубоких технических знаний. Диаграмму прецедентов можно изучить на рисунке 1.4.

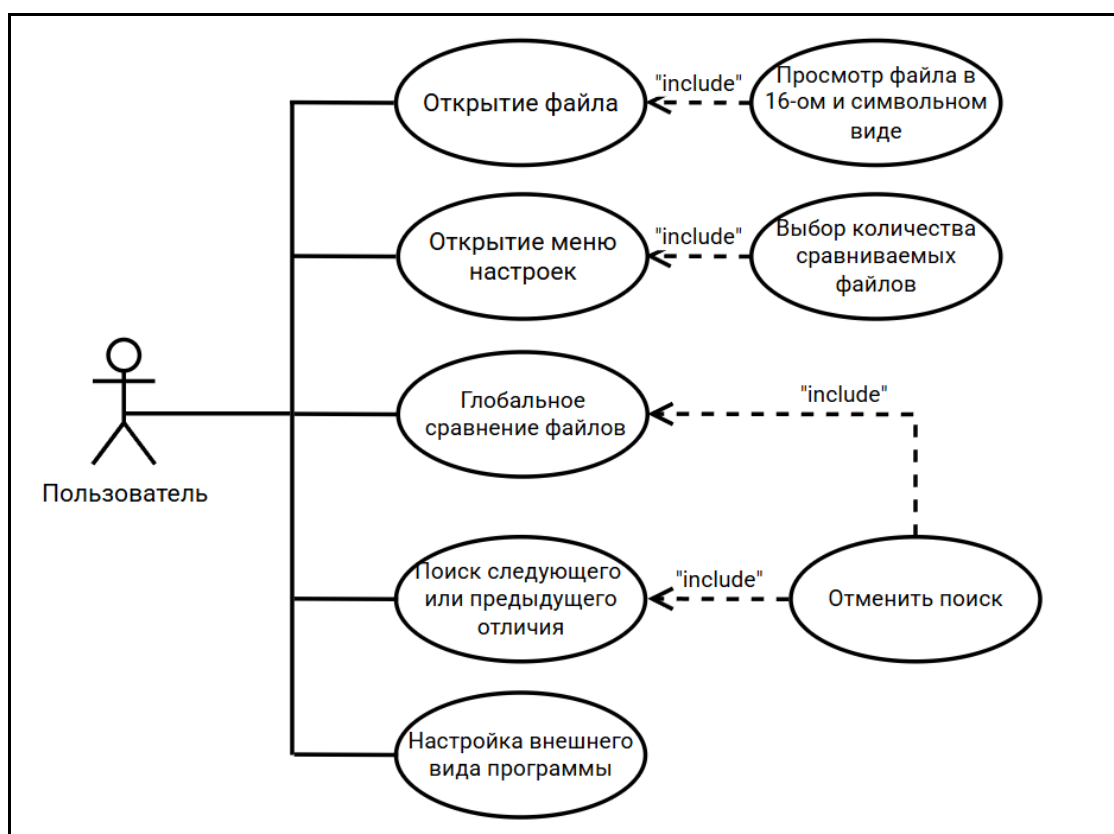


Рисунок 1.4 – UML-диаграмма прецедентов интерфейса пользователя

Модуль чтения и сравнения данных: Ядро программы, реализованное на C/C++ с использованием Windows API. Этот модуль отвечает за низкоуровневую работу с файлами, их побитовое сравнение и выявление различий. Для эффективной обработки файлов большого объема здесь используется механизм Memory-mapped files в сочетании с концепцией «скользящего окна». Данные читаются и сравниваются поблочно (по 64 КБ). Принцип «скользящего окна» позволяет подгружать в оперативную память только необходимые для текущей операции фрагменты файлов, значительно снижая потребление ОЗУ и обеспечивая высокую производительность даже при работе с терабайтами данных. Модуль также реализует алгоритм посимвольного перебора и сравнения, что обеспечивает точное выявление любых расхождений.

Модуль визуализации различий: Тесно связан с модулем UI. Он отвечает за преобразование информации об обнаруженных отличиях в наглядный для

пользователя вид. Для этого используется продуманная цветовая схема: каждому сравниваемому файлу присваивается свой уникальный цвет. Если блок данных в одном файле имеет схожесть с блоком в файле, расположенном справа от него, то схожая часть подсвечивается цветом изначального файла. Особый, красный цвет, используется для обозначения уникальных фрагментов данных, которые не имеют совпадений ни с одним другим сравниваемым файлом. Результаты отображаются с указанием смещения (оффсета), а также значений байтов в шестнадцатеричном (HEX) и символьном представлениях, предоставляя полную картину различий.

Взаимодействие между модулями происходит внутри единого приложения, что обеспечивает высокую скорость обмена данными и минимизирует задержки. Такая архитектура позволяет создать мощный и в то же время удобный инструмент для побитового сравнения файлов, способный эффективно решать задачи верификации и анализа данных любого объема.

1.6 Постановка задачи

Основная цель разработки – создание десктопного приложения, предназначенного для побитового сравнения множества файлов большого объема, что позволит обеспечить высокий уровень контроля за изменениями данных, сократить время на их верификацию и повысить качество анализа информации. Система призвана упростить процесс обнаружения даже мельчайших расхождений, автоматизируя рутинные задачи ручного поиска отличий и предоставляя детализированное, интуитивно понятное представление результатов.

Программа должна быть удобной и легкой в использовании, предоставляя пользователю возможность работать с файлами различных размеров – от одного килобайта до одного терабайта. Она должна обеспечивать высокую надежность выявления всех отличий на побитовом уровне и гарантировать безопасность обрабатываемых данных, не модифицируя исходные файлы. Для реализации этих задач система должна предоставлять возможность:

					ПГУ1.090301.10.001 ПЗ	Лист
						20
Изм.	Лист	№ докум.	Подпись	Дата		

- одновременного сравнения более двух файлов, что значительно расширяет аналитические возможности по сравнению с традиционными парными утилитами;
- быстрой навигации по обнаруженным отличиям с помощью функций перехода к следующей/предыдущей несоответствующей строке;
- глобальной оценки идентичности файлов, позволяющей моментально определить, являются ли все сравниваемые объекты полностью одинаковыми;
- детального и наглядного отображения различий, где каждый файл имеет свой уникальный цвет, схожие участки подсвечиваются цветом файла-источника, а абсолютно уникальные блоки данных выделяются контрастным красным цветом.

Эффективной обработки сверхбольших файлов за счет использования передовых методов, таких как Memory-mapped files, с интеллектуальным поблочным чтением данных (по 64 КБ) по принципу «скользящего окна», что минимизирует нагрузку на оперативную память и обеспечивает высокую производительность.

Кроме того, система должна быть оптимизирована для эффективной работы на операционной системе Windows, используя низкоуровневые возможности C/C++ и Windows API для достижения максимального быстродействия. В итоге, разрабатываемая программа должна стать мощным инструментом, который повысит эффективность работы специалистов, занимающихся верификацией данных, анализом целостности информации и контролем версий, обеспечивая новый уровень детализации и удобства в этой предметной области.

1.7 Выбор инструментов разработки

Правильный выбор инструментов разработки является одним из ключевых факторов при создании высокопроизводительного программного обеспечения. Он позволяет оптимизировать процесс разработки, сократить время и затраты на создание продукта, а также обеспечить качество и безопасность разработки. Неправильный выбор инструментов может привести к проблемам в процессе

					<i>ПГУ1.090301.10.001 ПЗ</i>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		21

разработки, а также к потенциальным проблемам при эксплуатации готового продукта. Поэтому критически важно правильно подойти к выбору инструментов разработки, учитывая конкретные требования и характеристики проекта, особенно при работе с низкоуровневыми операциями и большими объемами данных.

1.7.1 Выбор среды разработки

Выбор интегрированной среды разработки (IDE) существенно влияет на скорость, удобство и эффективность процесса создания программного обеспечения. Современные IDE предоставляют широкий набор функций, таких как автодополнение кода, отладка, интеграция с системами контроля версий и поддержка различных языков программирования.

Для разработки программы побитового сравнения файлов на языке C/C++ был выбран Microsoft Visual Studio 2019. Это одна из наиболее мощных и полнофункциональных IDE, предлагающая обширный набор инструментов для разработки приложений под операционную систему Windows. Среди ключевых преимуществ Visual Studio 2019 применительно к данной задаче можно выделить:

Полная поддержка C/C++ и нативного WinAPI: IDE предоставляет высококачественные компиляторы, библиотеки и отладчики, оптимизированные для работы с C/C++ и непосредственного использования функций Windows API. Это критически важно для реализации пользовательского интерфейса и всей логики программы без использования высокоуровневых фреймворков.

Развитые средства отладки: Интегрированный отладчик Visual Studio является одним из лучших в отрасли, позволяя эффективно выявлять и устранять ошибки в сложном системном коде и низкоуровневых операциях с памятью.

Интеграция с нативными Windows-приложениями: visual Studio идеально подходит для разработки классических десктопных приложений, управляемых сообщениями Windows, предоставляя все необходимые шаблоны проектов и инструменты.

Обширное сообщество и документация: наличие огромного сообщества разработчиков и исчерпывающей документации по Visual Studio и WinAPI обеспечивает легкий доступ к помощи и ресурсам при возникновении вопросов.

Проектная структура: visual Studio упрощает управление сложными проектами, предоставляя удобную структуру для организации исходного кода, ресурсов и библиотек.

Эти преимущества делают visual Studio 2019 оптимальным выбором для проекта, требующего глубокой интеграции с операционной системой и максимальной производительности, достигаемой за счет прямого взаимодействия с WinAPI.

1.7.2 Выбор языка программирования

Выбор языка программирования является фундаментальным решением, определяющим архитектуру, производительность и сложность разработки. Каждый язык обладает уникальными особенностями, которые существенно влияют на процесс создания и конечный результат.

Для реализации программы побитового сравнения файлов был выбран C/C++. Этот выбор обусловлен следующими ключевыми факторами:

Низкоуровневый доступ и производительность: C/C++ позволяют работать с данными на уровне байтов и битов, предоставляя максимальный контроль над аппаратными ресурсами. Компилируемый характер языка обеспечивает высочайшую скорость выполнения, что критически важно для эффективной обработки файлов внушительных размеров.

Прямое взаимодействие с Windows API: C/C++ являются основным языком для прямого использования функций Windows API, что позволяет реализовать пользовательский интерфейс, файловые операции и управление памятью на максимально низком уровне. Это исключает накладные расходы, связанные с использованием управляемых сред или высокоуровневых фреймворков.

Управление памятью: C/C++ предоставляют прямой контроль над памятью, что позволяет реализовать оптимизированные стратегии, такие как Memory-

mapped files и «скользящее окно», минимизируя потребление оперативной памяти даже при сравнении очень больших файлов.

Надежность и стабильность: При правильном использовании, C/C++ позволяют создавать очень надежные и стабильные приложения, что важно для инструмента, предназначенного для верификации данных.

Другие языки программирования, такие как C#, Python или Java, хотя и обладают своими преимуществами в области скорости разработки и кроссплатформенности, уступают C/C++ в задачах, требующих максимальной производительности, низкоуровневого контроля над системными ресурсами и прямого взаимодействия с WinAPI для создания чистых нативных приложений.

1.7.3 Выбранные технологии

В ходе проведенного анализа и обоснования, для разработки программы побитового сравнения файлов были приняты следующие основные технологические решения:

Среда разработки: Microsoft Visual Studio 2019.

Язык программирования: C/C++.

API для работы с ОС и UI: Нативный Windows API (WinAPI), используемый для реализации логики программы, файловых операций и построения графического пользовательского интерфейса.

1.8 Реализация

1.8.1. Создание проекта

Перед началом написания программы на C++ необходимо установить программу позволяющую работать с данным языком. Нужно перейти на официальный сайт «<https://visualstudio.microsoft.com/ru/vs/older-downloads/>» и скачать бесплатную и полнофункциональную среду Visual Studio Community 2019 как показано на рисунке 1.5.

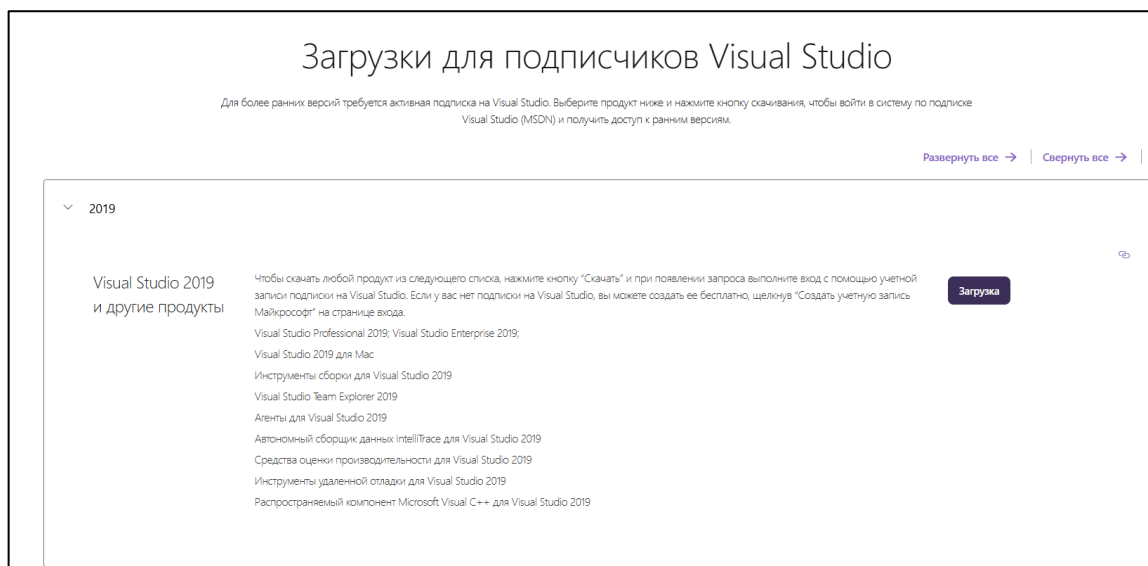


Рисунок 1.5 – Страница загрузки установщика Visual Studio 2019

После завершения загрузки запускаем данный установщик. В открывшемся окне нам необходимо выбрать в качестве компетентна «Разработка классических приложений на C++» как показано на картинке 1.6.

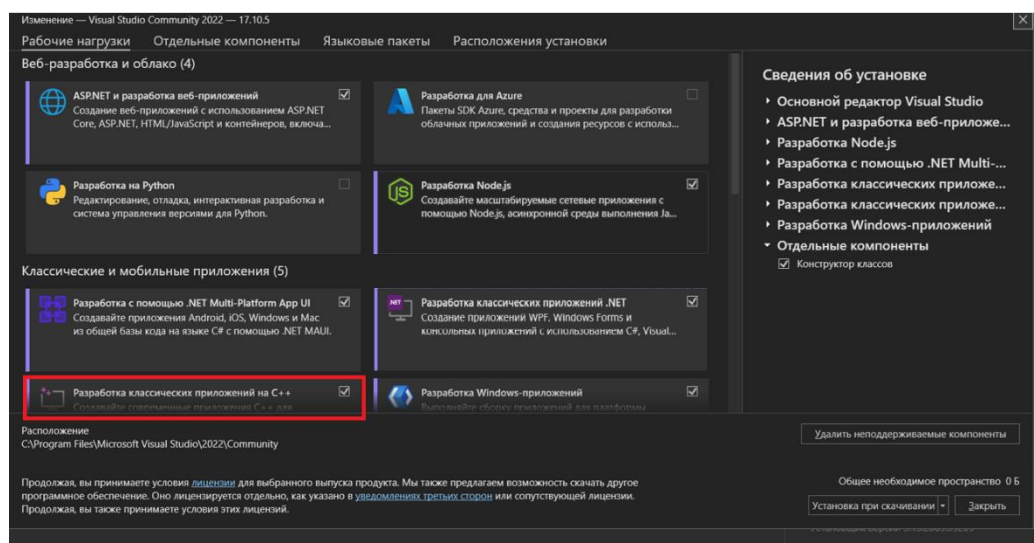


Рисунок 1.6 – Выбор компонентов для установки

Далее необходимо создать проект, в котором будем разрабатывать наше приложение, переходим в установленный ранее Visual Studio 2019 и выбираем пункт «Создание проекта» как показано на рисунке 1.7.

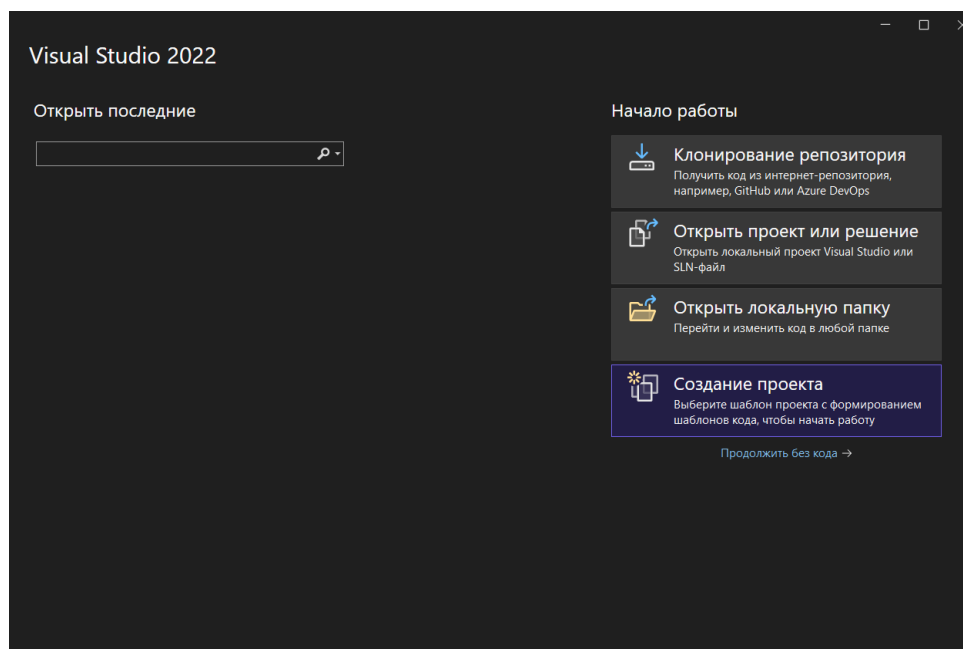


Рисунок 1.7 – Меню создание проекта

Следующем шагом в качестве типа создаваемого проекта указываем Empty project (Пустой проект), как показано на рисунке 1.8.

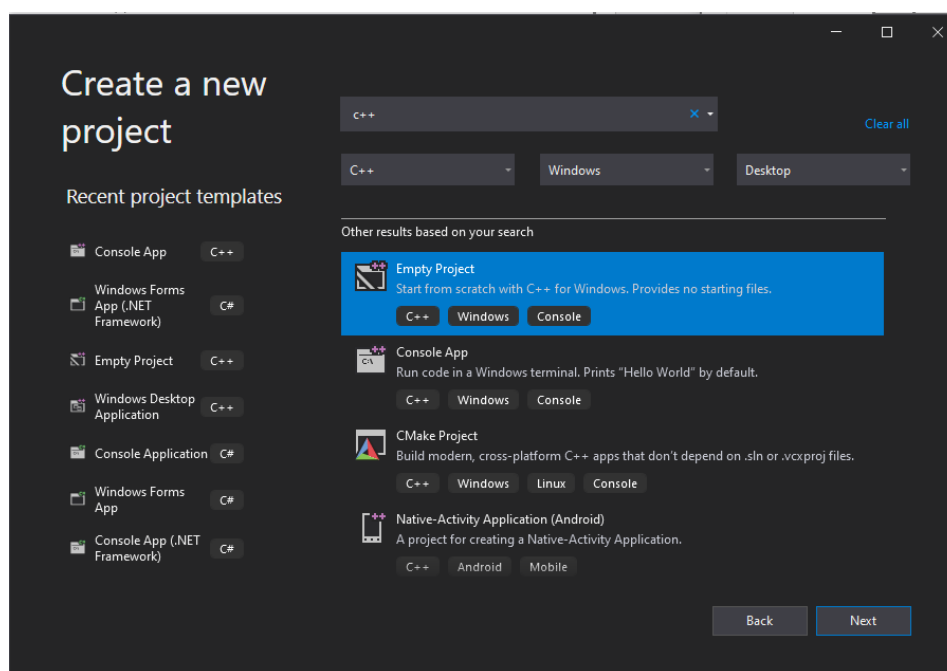


Рисунок 1.8 – Меню создание проекта

Далее указываем имя проекта и каталог, где будет располагаться проект, как показано на рисунке 1.9.

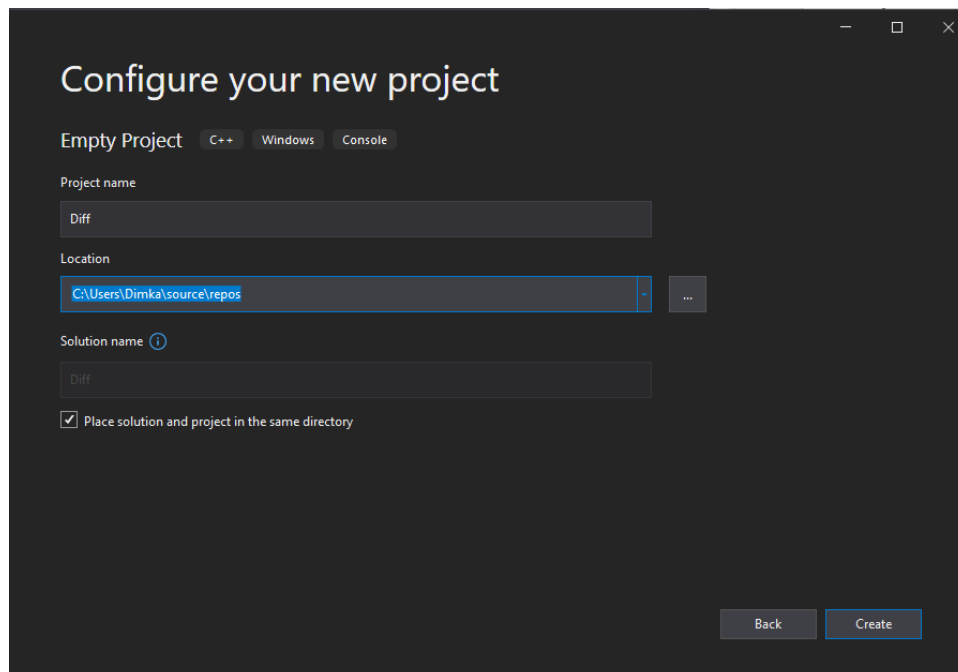


Рисунок 1.9 – Меню создание проекта

После этих действий будет создан проект для создания приложений на C/C++, в котором мы можем приступить к разработке нашего приложения программы для побитового сравнения файлов, как показано на рисунке 1.10.

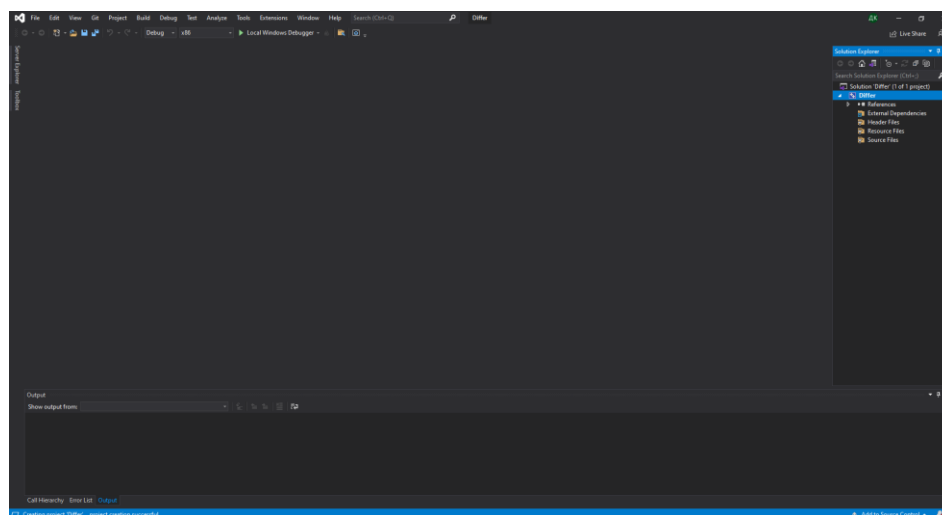


Рисунок 1.10 – Окно редактора программы

1.8.2. Описание доступных функций

Программа Diff предполагает реализацию таких функций, как:

- открытие файлов любого формата;

					ПГУ1.090301.10.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		27

- отображение файла в символьном (оригинальном) виде;
- отображение файла в шестнадцатеричном виде;
- быстрое и удобное перемещение по файлу, вне зависимости от размера;
- визуальное отображение отличий и схожестей;
- навигация по обнаруженным отличиям;
- глобальное сравнение файлов;
- настройка визуальных параметров программы;
- сохранение настроек визуальных параметров программы в реестр.

Для начала работы с программой ее нужно открыть, для этого достаточно просто сделать двойное нажатие мышкой по иконке программы, как показано на рисунке 1.11.

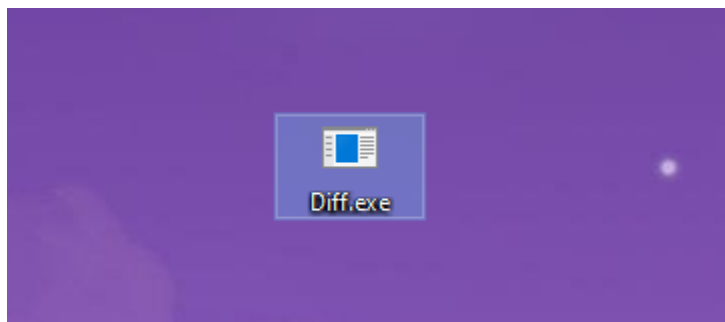


Рисунок 1.11 – Иконка программы

После открытия программы, мы попадаем в главное окно, представленное на рисунке 1.12.

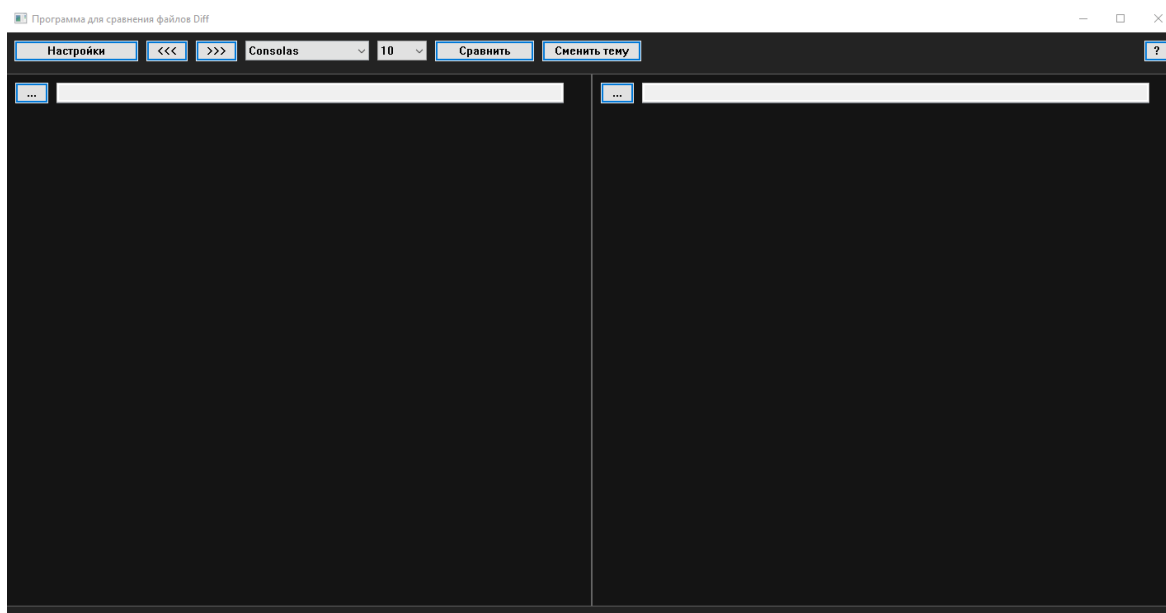


Рисунок 1.12 – Главное окно программы

По стандарту при первом запуске программы, количество файлов для сравнения выбрано два. Можно открывать файлы с помощью соответствующих кнопок «...», справа от этих кнопок находится полоса отображения пути и имени открытого файла. Как выглядит кнопка и полоса показано на рисунке 1.13.



Рисунок 1.13 – Кнопка открытия файла полоса с именем файла

После нажатия на кнопку открытия файлов «...», открывается стандартное окно проводника Windows для выбора нужного файла. Окно выбора файла представленно на рисунке 1.14.

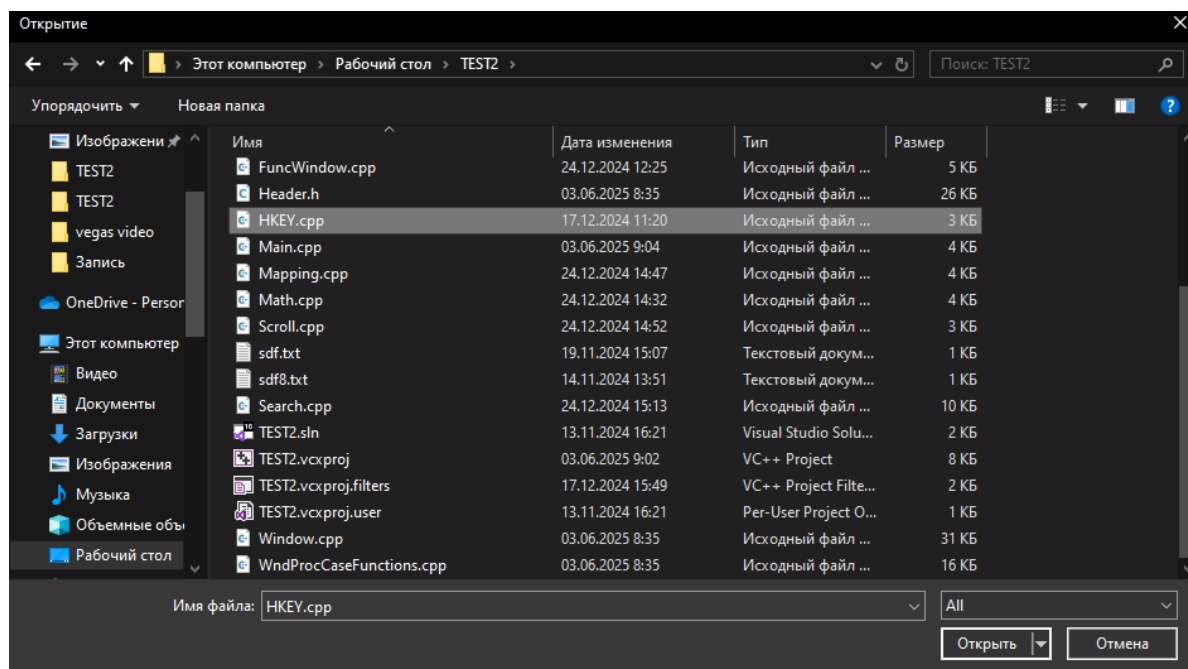


Рисунок 1.14 – Окно выбора файла

Для выбора файла можно выделить файл и нажать кнопку открыть, либо сразу открыть файл двойным нажатием левой кнопки мыши. После выбора файла, появится символьное и шестнадцатеричное отображение файла, в соответствующем окне. На одной строчке всегда отображается 16 символов. Также слева от файла отображается адрес (оффсет смещения) в файле. Итоговый вид отображения файла изображен на рисунке 1.15.

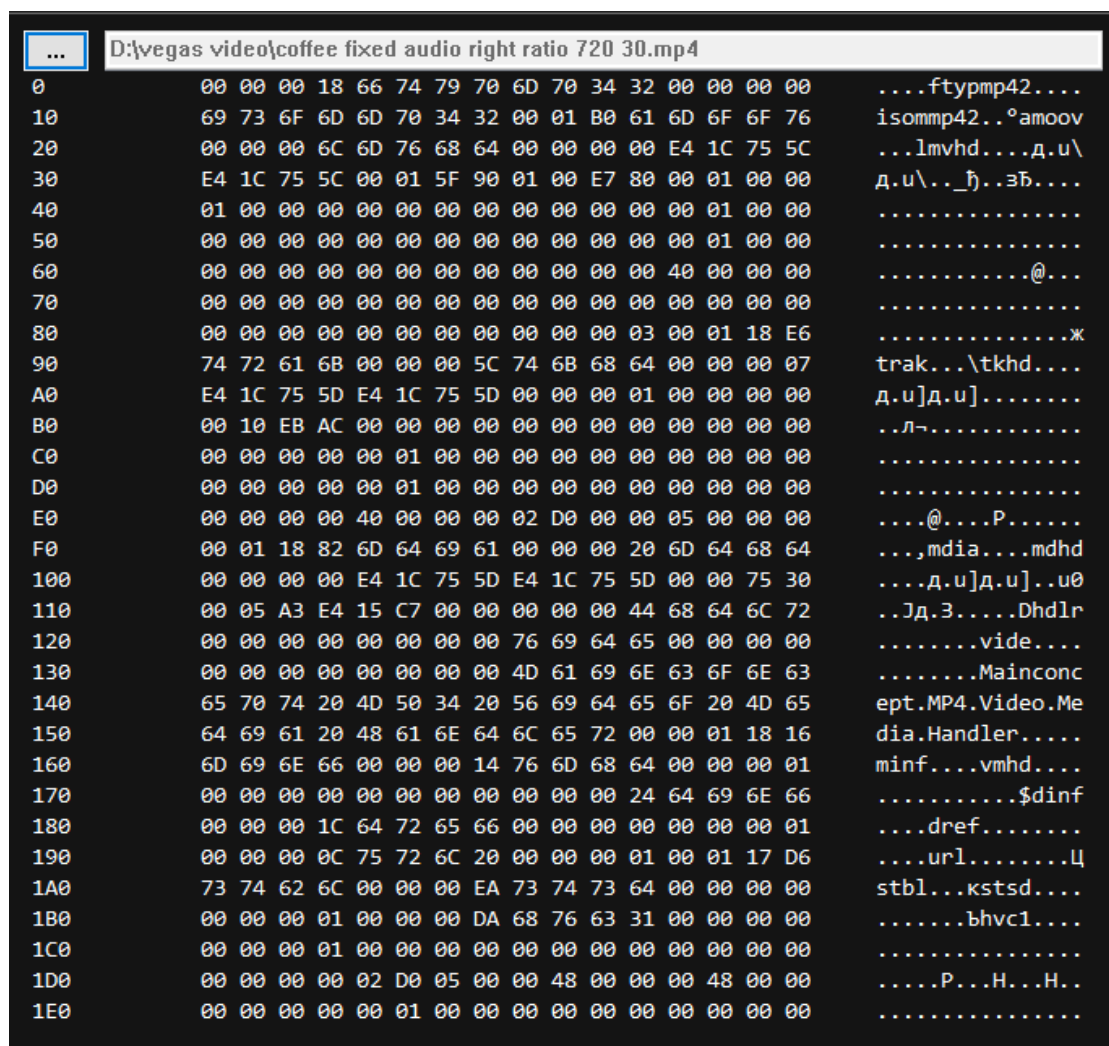


Рисунок 1.15 – Стандартное отображение файла в программе

В приложении, для перемещения по файлу, заложены стандартные функции, такие как:

- скролл (прокрутка);
- построчное перемещение с помощью стрелочек на клавиатуре или на самом скролле (вверх\вниз);
- постраничное перемещение с помощью клавиш pgup и pgdn;
- перемещение с помощью колесика мыши, одно движения колесика перемещает на 4 строчки.

Для того чтобы просматривать файл имеются вертикальный и горизонтальный скроллы. Каждый скролл в автоматическом режиме отключается,

если он не нужен в данной ситуации – вертикальный, если файл меньшего размера чем окно программы, горизонтальный если отображение файла по ширине меньше чем ширина окна. Пример отображения всех скроллов можно увидеть на рисунке 1.16.

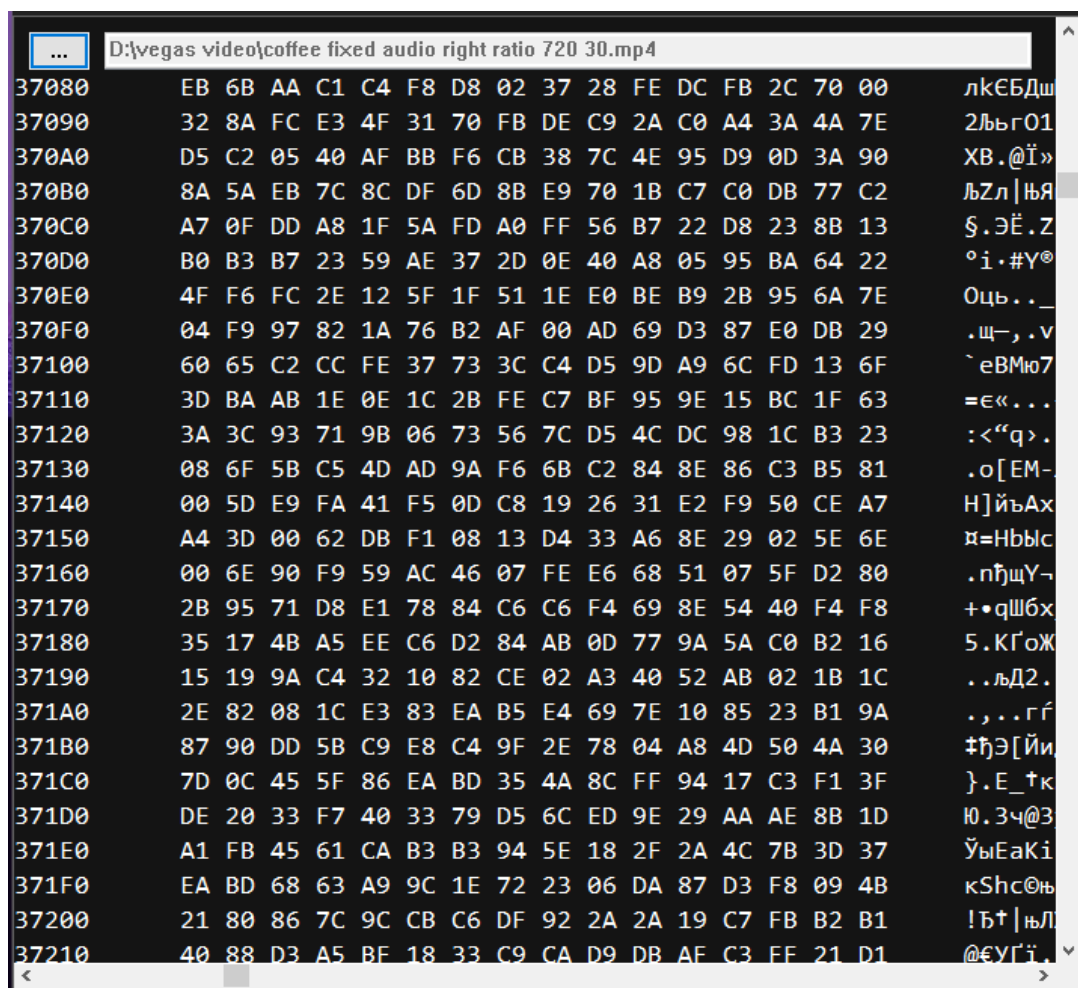


Рисунок 1.16 – Отображение горизонтального и вертикального скроллов

Пример вида программы без всех скроллов (прокруток) можно увидеть на рисунке 1.17.

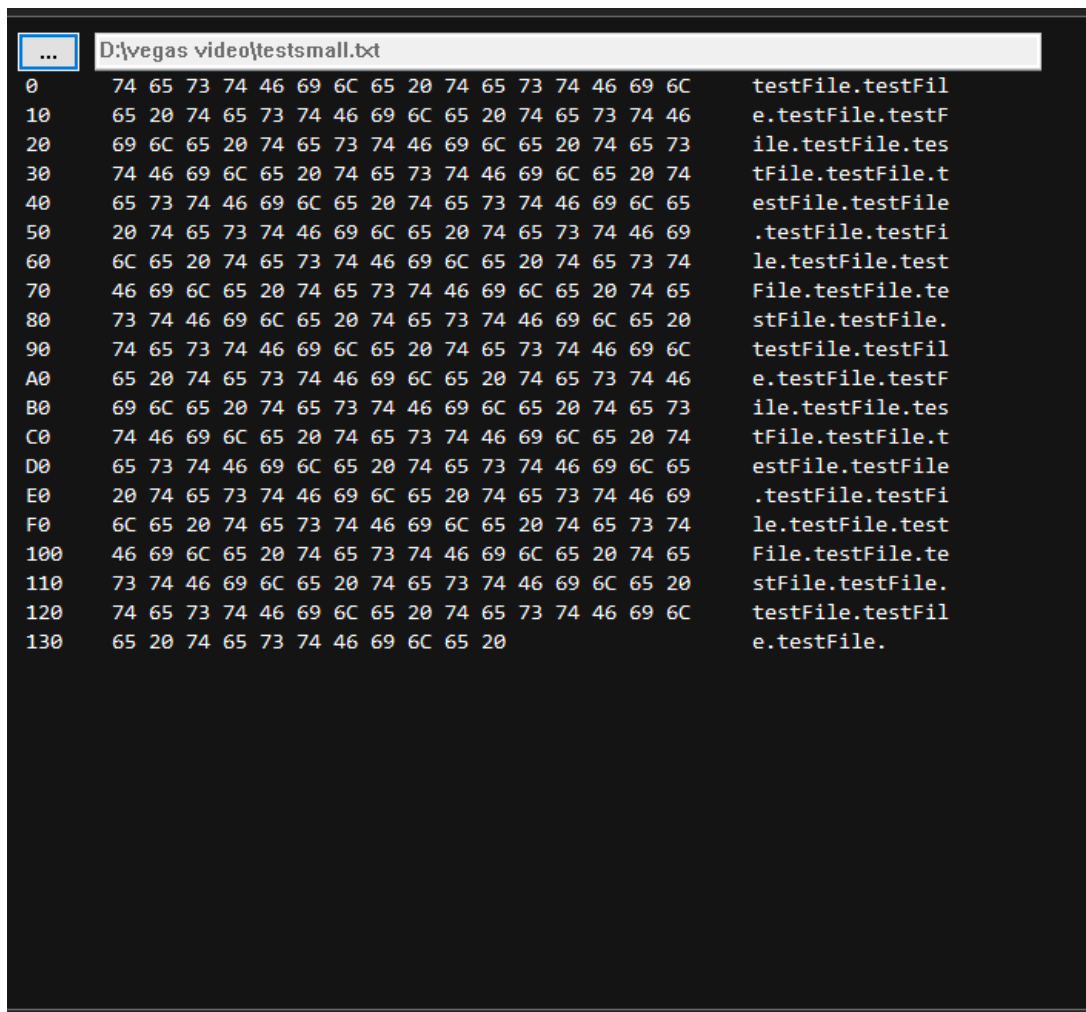


Рисунок 1.17 – Отображение программы без горизонтального и вертикального скроллов

При открытии второго и больше файлов, автоматически включается режим отображения различий. Красным цветом показаны отличия, которые не повторяются ни в одном файле. Пример отображения отличий в процессе работы программы можно увидеть на рисунке 1.18.

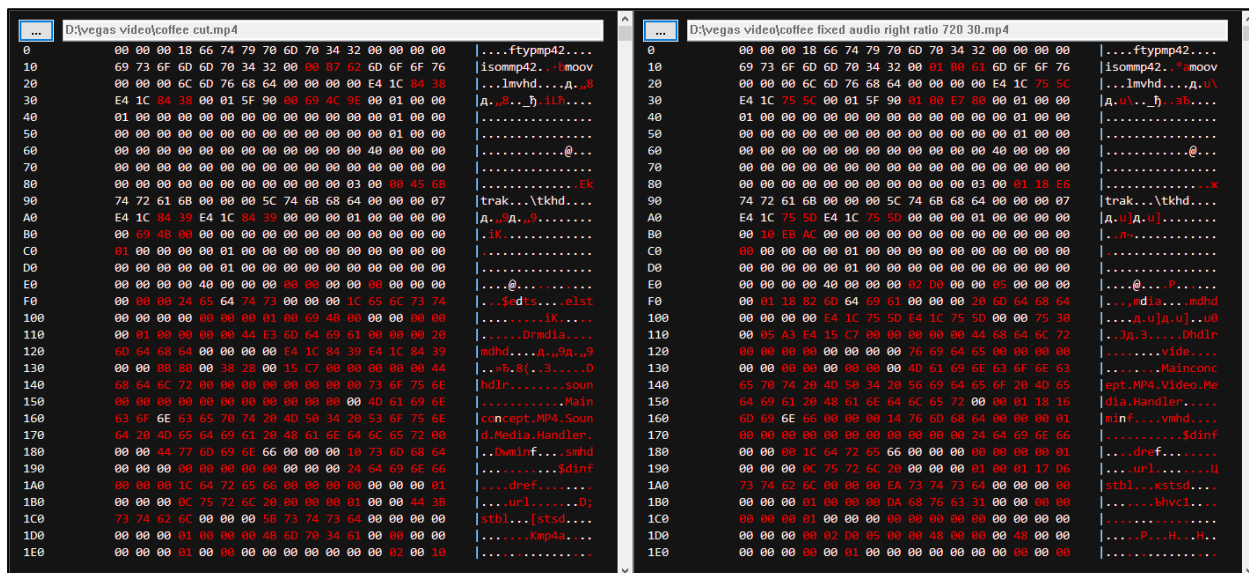


Рисунок 1.18 – Отображение отличий между файлами

Для информативного и читаемого отображения отличий при сравнении более двух файлов, был создан алгоритм, основанный на том подходе, что при таком количестве файлов легче показывать схожести. Работает алгоритм так, у каждого файла есть свой цвет, он отвечает за схожести с другими файлами, например у первого файла белый цвет, у второго салатовый и т.д.. Сравнение идет слева направо, соответственно если все схожести с первым файлом во всех других будут отображаться белым. Все символы похожие со вторым файлом, но не похожие с первым будут отображаться зеленым. Пример работы с несколькими файлами можно увидеть на рисунках 1.19 и 1.20, разница между рисунками в размере шрифта.

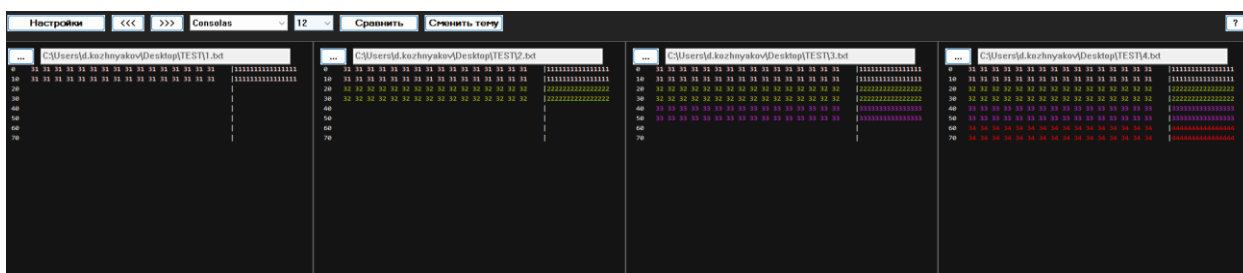


Рисунок 1.19 – Пример работы с 4 файлами, размер шрифта 12

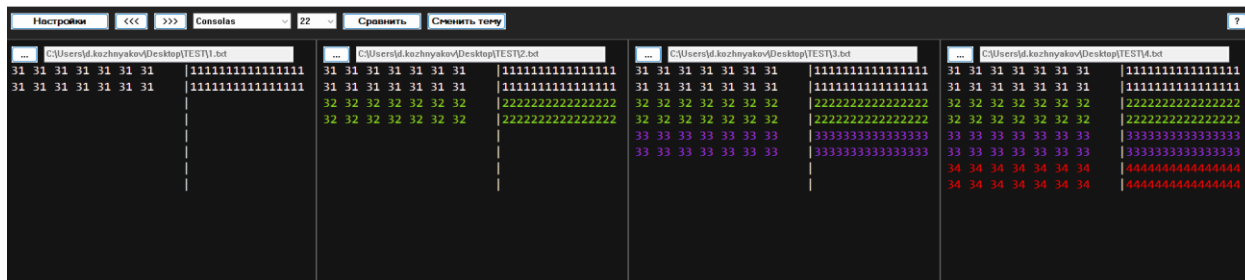


Рисунок 1.20 – Пример работы с 4 файлами, размер шрифта 22

Была реализована навигация для поиска следующего или предыдущего отличия, программа при поиске может пропускать блоки полностью одинаковых или полностью различных символов, для удобства работы с различиями. То как выглядят кнопки поиска отличий можно увидеть на рисунке 1.21.

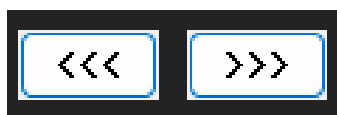


Рисунок 1.21 – Кнопки поиска и перехода к отличиям

Пример работы кнопки поиска отличий вперед можно увидеть на рисунках 1.22 и 1.23. На рисунках изображены окна до нажатия кнопки и после нажатия.

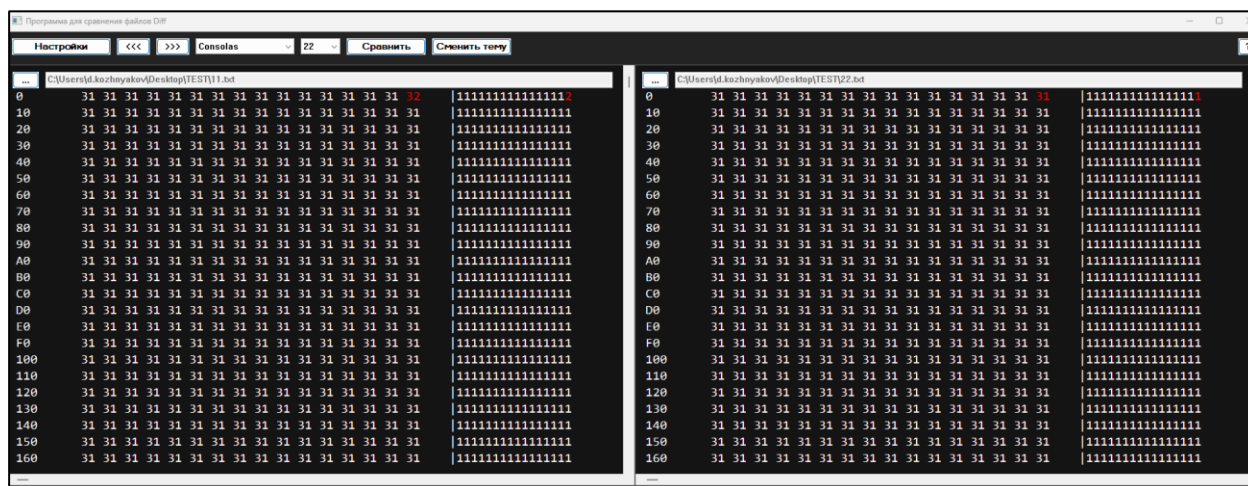


Рисунок 1.22 – Вид программы до нажатия кнопки поиска следующего отличия

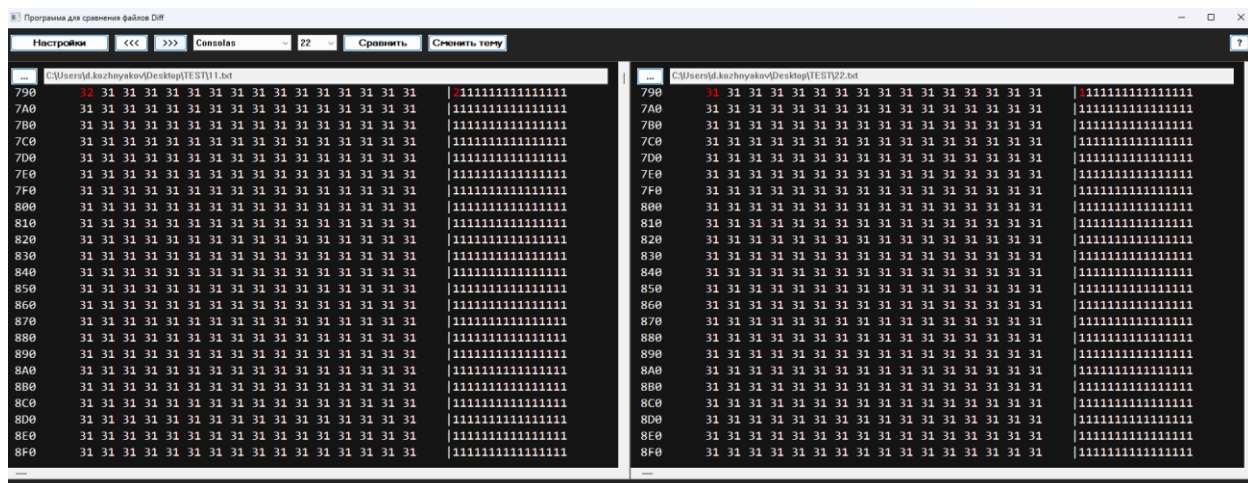


Рисунок 1.23 – Вид программы после нажатия кнопки поиска
следующего отличия

В программе присутствует функционал глобального сравнения файлов. В это режиме программа запускает цикл полного сравнения файлов и по итогам сообщает о идентичности, либо различия файлов. Пример итоговых сообщений можно увидеть на рисунках 1.24 и 1.25.

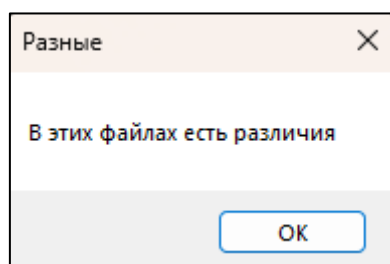


Рисунок 1.24 – Сообщение о различиях файлов

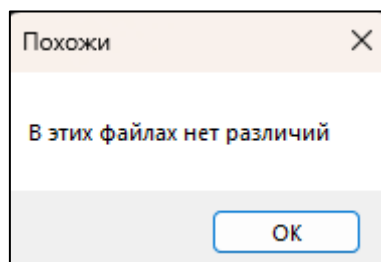


Рисунок 1.25 – Сообщение о схожести файлов

Во время глобального сравнения файлов или поиска отличий, в программе отображается прогресс поиска, его представление можно увидеть на рисунке 1.26.

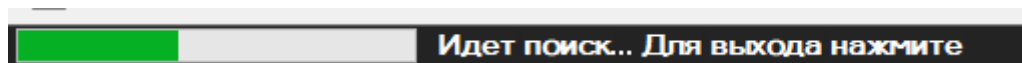


Рисунок 1.26 – Отображение прогресса поиска отличий

Чтобы прекратить поиск есть несколько способов: можно нажать на кнопку Esc и пользователю предложится прекратить поиск, окно о прекращении поиска можно увидеть на рисунке 1.27, также можно попытаться закрыть программу во время поиска или попытаться открыть новый файл, все эти действия приведут к сообщению о прекращении поиска.

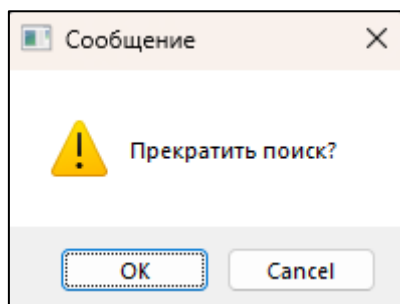


Рисунок 1.27 – Сообщение о прекращении поиска

Для удобства пользователя в программе присутствуют настройки внешнего вида программы, такие как смена шрифта, изменение размера шрифта и изменение цвета темы (черная/белая), представлены на рисунке 1.28.

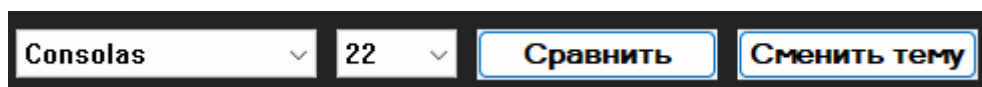


Рисунок 1.28 – Настройки внешнего вида программы

На рисунке 1.29 можно увидеть, как выглядит белая тема программы.

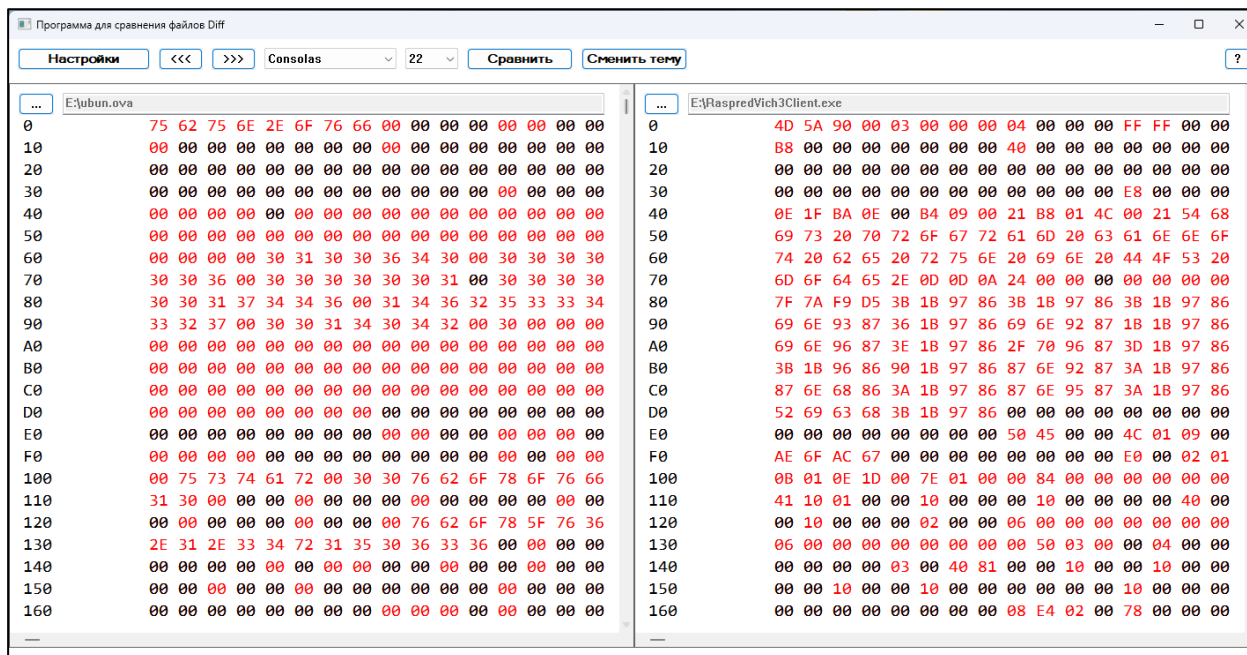


Рисунок 1.29 – Вид программы с белой темой

Чтобы выбрать нужное количество сравниваемых файлов, нужно зайти в меню настроек с помощью кнопки «Настройки». Внутри меню, представленного на рисунке 1.30, можно выбрать нужное количество файлов из выпадающего списка. Чтобы применить изменения нужно нажать кнопку «Продолжить» либо чтобы не применять изменения нажать на крестик или кнопку «Отменить».

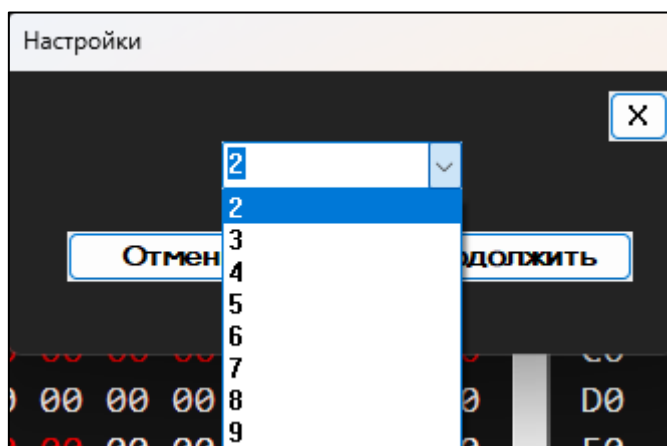


Рисунок 1.30 – Меню настроек

Приложение Diff сохраняет все настройки внешнего вида и количества файлов. Поэтому при повторном запуске программы не надо настраивать все заново. Сохранение настроек происходит в реестр, в папку HKEY_CURRENT_USER\Software\Diff. Как выглядят настройки в реестре показано на рисунке 1.31.

Name	Type	Data
 (Default)	REG_SZ	(value not set)
 fontName	REG_SZ	Consolas
 fontSize	REG_DWORD	0x00000016 (22)
 NumOfFiles	REG_DWORD	0x00000002 (2)
 Theme	REG_SZ	BLACK

Рисунок 1.31 – Сохранение настроек в реестр

1.8.3. Описание программы

Проект организован модульно для обеспечения чистоты кода, упрощения разработки и поддержки. Каждый функциональный блок реализован в отдельном файле или паре файлов (заголовочный файл .h и файл реализации .cpp). Основные компоненты и их реализация распределены следующим образом:

- main.cpp: содержит главную точку входа в приложение (winmain). отвечает за регистрацию основных классов окон (главного, дочерних для файлов, окна настроек), инициализацию глобальных переменных и запуск основного цикла обработки сообщений windows;

- header.h: общий заголовочный файл, содержащий глобальные константы (id элементов управления, размеры буферов), объявления внешних переменных, структуры данных (например, для маппинга файлов) и прототипы всех функций, используемых в различных модулях;

- funcwindow.cpp: реализует вспомогательные функции, связанные с управлением окнами и элементами интерфейса. включает логику для динамического изменения количества окон для файлов, расчет доступного

размера для отображения строк и общую функцию обновления (перерисовки) окон;

– `mapping.cpp`: содержит функции для работы с механизмом `memory-mapped files`. здесь реализованы создание и закрытие отображений файлов в память, а также функции для отображения конкретных частей файла (окна) с заданным смещением;

– `math.cpp`: включает математические и вспомогательные функции, такие как определение максимального размера среди открытых файлов, проверки вхождения числа в диапазон, а также функции для преобразования байтов в шестнадцатеричное представление (`chartohex`) и их корректного отображения в символьном виде;

– `scroll.cpp`: отвечает за всю логику, связанную с полосами прокрутки. содержит функции для определения необходимости горизонтальной и вертикальной прокрутки, установки позиции ползунка и расчета смещения данных в файле на основе положения прокрутки;

– `search.cpp`: реализует основной алгоритм поиска отличий между файлами. содержит функции для запуска процесса сравнения в отдельном потоке (`startdiffthread`) для обеспечения отзывчивости интерфейса, а также логику пошагового поиска различий (`searchcycle`) и управления прогрессом сравнения;

– `hkey.cpp`: предоставляет функционал для взаимодействия с реестром `windows`. используется для чтения и сохранения настроек программы (например, выбранных шрифтов, количества файлов, цветовой темы), что позволяет сохранять пользовательские предпочтения между запусками;

– `window.cpp`: содержит оконные процедуры (`windowproc`) для различных окон, включая главное окно и окно настроек. обрабатывает сообщения `windows`, управляющие поведением `ui`, такие как нажатия кнопок, выбор элементов из списков, изменение размера окон и сообщения `wm_paint` для отрисовки;

– `wndproccasefunctions.cpp`: вспомогательный файл, содержащий отдельные функции-обработчики для различных сообщений `windows`, которые вызываются

из основных оконных процедур. например, здесь реализована логика изменения цветовой темы, а также обработка нажатий клавиш для навигации и управления прокруткой.;

На рисунке 1.32 можно увидеть UML диаграмму деятельности. Также на рисунке 1.33 представлена диаграмма компонентов. Рисунок 1.34 демонстрирует диаграмму последовательности.

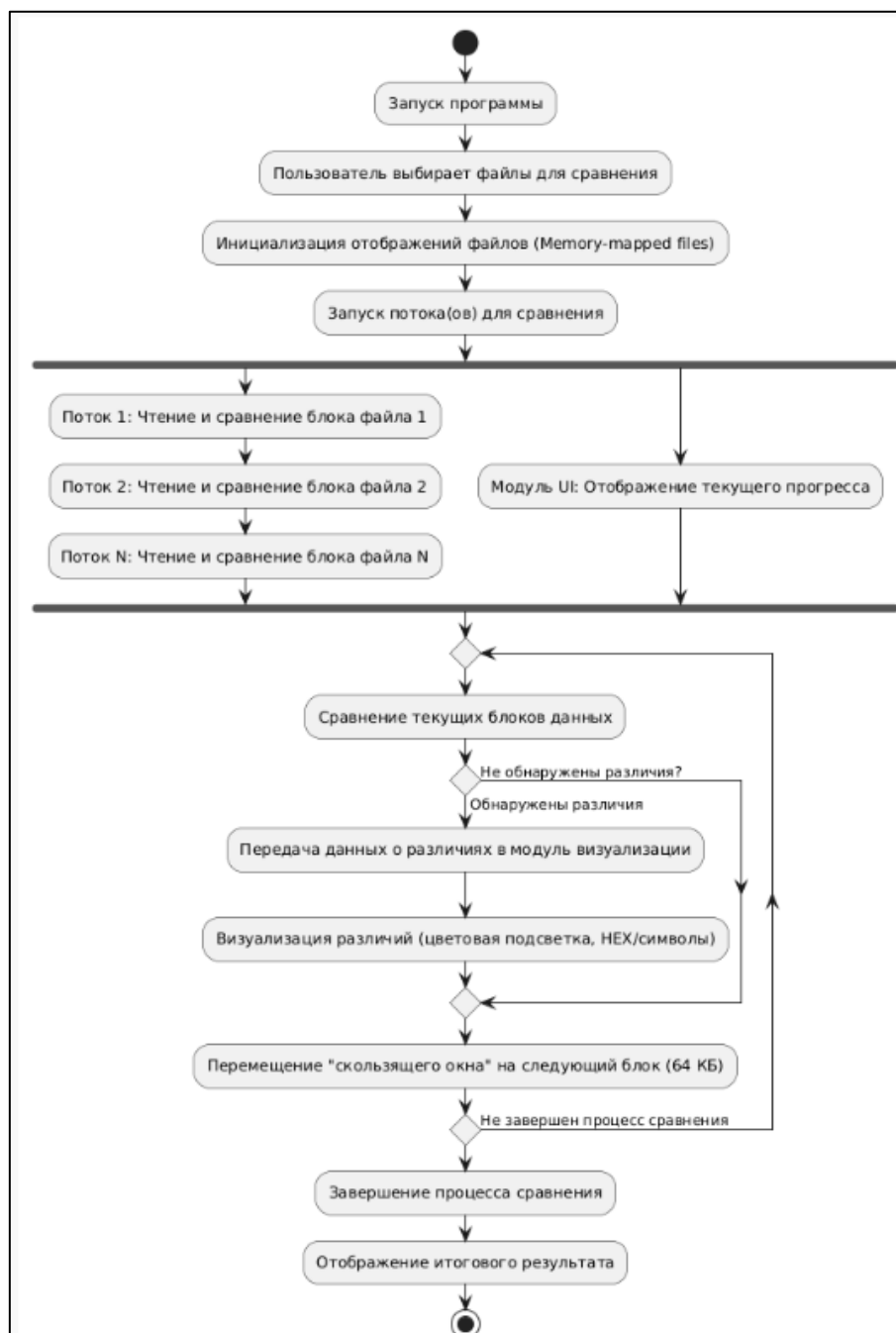


Рисунок 1.32 – UML- диаграмма деятельности

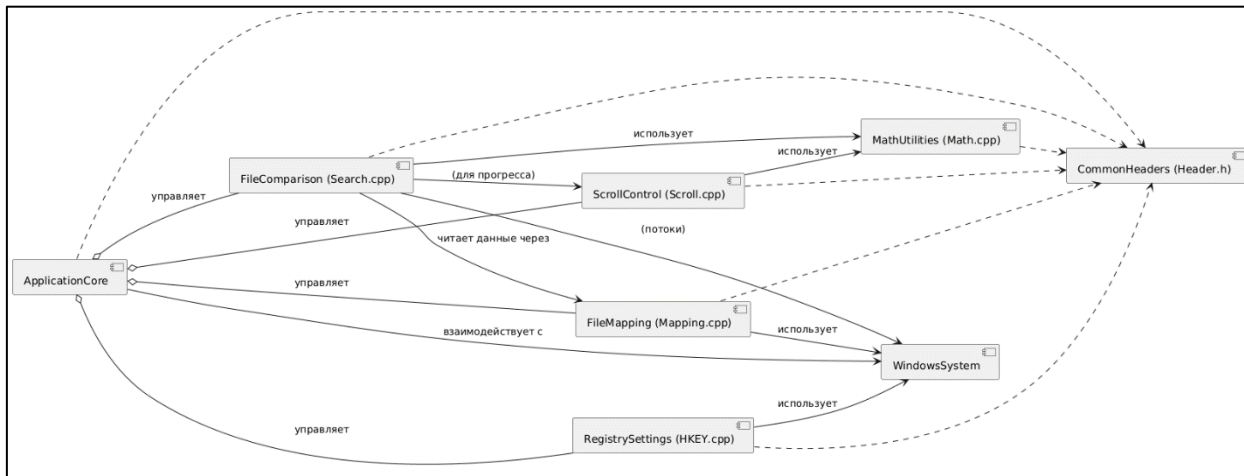


Рисунок 1.33 – UML-диаграмма компонентов

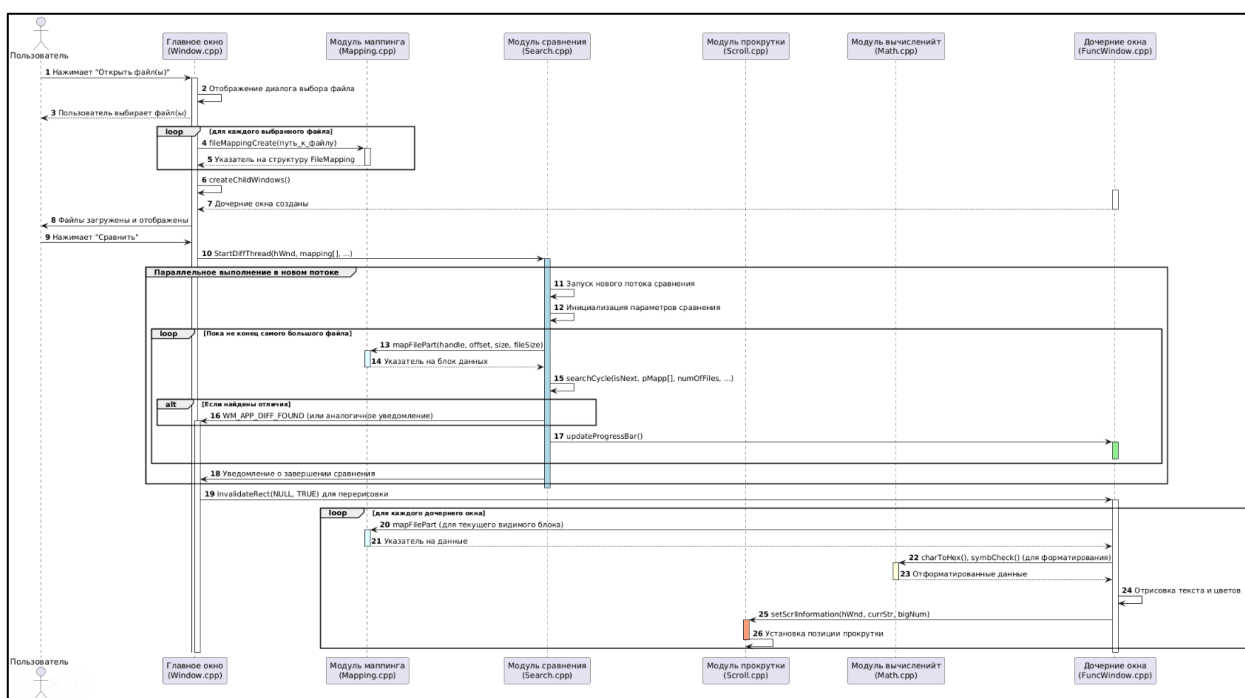


Рисунок 1.34 – UML-диаграмма компонентов

2 Безопасность жизнедеятельности

2.1 Правила безопасности при использовании ПК

Данный раздел составлен с учетом требований ГОСТ 12.0.003-2015 «Система стандартов безопасности труда. Электронно-вычислительная техника. Общие требования безопасности» и направлен на выявление и минимизацию возможных негативных воздействий. Ниже приведен перечень основных рисков, которым следует придать особое значение:

– риск повреждения зрения и утомления глаз:

1) длительное использование ПК без перерывов и отдыха глаз;

2) плохая эргономика рабочего места, включая неправильное расположение монитора, отражение света на экране и недостаточное освещение;

– риск травм и заболеваний опорно-двигательной системы:

1) неправильное положение тела при работе с ПК, например, неправильная осанка, неправильная поза или неправильное расположение рук и ног;

2) отсутствие эргономической мебели и принадлежностей, таких как стулья с подлокотниками и подставки для рук;

– риск воздействия электромагнитных полей:

1) использование устаревшего или неисправного оборудования, которое может создавать сильные электромагнитные поля;

2) неправильное расположение оборудования, например, слишком близко к рабочему месту или другим электронным устройствам;

– риск пожара и электрического удара:

1) неправильное подключение и использование электрооборудования, включая перегрузку розеток и использование поврежденных кабелей;

2) нарушение правил пожарной безопасности, включая использование запрещенных электроприборов вблизи ПК и наличие горючих материалов рядом с оборудованием;

– риск нарушения информационной безопасности:

1) небезопасное использование паролей и учетных записей, включая слабые пароли или их совместное использование;

2) несанкционированный доступ к компьютерной информации, включая утечку конфиденциальных данных;

– риск воздействия шума: использование шумных компьютерных компонентов или работа в шумной среде.

Соблюдение правил техники безопасности и эргономических рекомендаций позволяет значительно снизить указанные риски и обеспечить комфортные и безопасные условия труда при работе с персональным компьютером.

					ПГУ1.090301.10.001 ПЗ	Лист
						44
Изм.	Лист	№ докум.	Подпись	Дата		

3 Стартап проект

Данный проект направлен на разработку программного обеспечения для высокопроизводительного и точного сравнения файлов больших объемов на побитовом уровне. Программа будет предназначена для эффективной верификации целостности данных, выявления модификаций и синхронизации копий файлов, где даже незначительные расхождения могут иметь критические последствия.

Разрабатываемое приложение будет обладать рядом преимуществ по сравнению с существующими аналогами, таких как:

- поддержка сравнения файлов размером от нескольких килобайт до 1 терабайта;

- детальное отображение результатов сравнения, включая смещения, шестнадцатеричные значения байтов и контекст различий;

- высокая производительность и эффективное использование системных ресурсов за счет применения низкоуровневых операций ввода-вывода и техник работы с памятью;

разработанное приложение найдет широкое применение в областях, требующих максимальной точности при работе с большими данными, таких как архивация, резервное копирование, криминалистическая экспертиза и многие другие.

3.1 Сегменты ЦА

Конечным продуктом, полученным в результате выполнения данной работы, является программное обеспечение для побитового сравнения файлов большого объема.

Разрабатываемое приложение будет удовлетворять потребность пользователей в надежном и производительном инструменте для верификации целостности данных, где даже минимальные расхождения могут иметь

					ПГУ1.090301.10.001 ПЗ	Лист
						45
Изм.	Лист	№ докум.	Подпись	Дата		

критические последствия. Программа будет решать проблему ограниченности существующих решений при работе с файлами гигабайтных и терабайтных размеров, предоставляя эффективные алгоритмы сравнения и детальное отображение результатов.

Целевыми потребителями разработанного ПО будут предприятия и организации, работающие с критически важными данными, такие как:

- центры обработки данных и поставщики услуг облачного хранения;
- правоохранительные органы и судебно-экспертные учреждения;
- научно-исследовательские институты и лаборатории;
- производственные предприятия, использующие системы контроля качества.

Продвижение продукта будет осуществляться через прямые продажи, работу с дистрибьюторами, а также участие в тематических выставках и публикации статей в отраслевых СМИ.

3.2 Ключевые процессы

Для успешного запуска, функционирования и масштабирования стартап-проекта по разработке программного обеспечения для побитового сравнения файлов необходимо четко определить и организовать ключевые процессы. Эффективное управление этими процессами позволит минимизировать риски, оптимизировать использование ресурсов и обеспечить достижение поставленных бизнес-целей.

Ниже перечислены основные ключевые процессы, необходимые для реализации данного стартап-проекта:

- разработка программного обеспечения;
- маркетинг и продажи продукта;
- внедрение и поддержка продукта;
- управление проектом и бизнес-процессами.

Разработка программного обеспечения является центральным процессом данного стартап-проекта. Он включает в себя все этапы создания программного продукта, начиная от детального проектирования архитектуры и модулей, выбора оптимальных алгоритмов, до непосредственного написания. Процесс разработки также включает тестирование функциональности, производительности и надежности программы на различных типах и объемах файлов. Подробное описание этапов разработки ПО представлено в разделе 1.8 данной выпускной квалификационной работы. Непрерывное совершенствование продукта, добавление новых функций и оптимизация существующих являются частью этого процесса.

Маркетинг и продажи продукта направлены на привлечение целевой аудитории и конвертацию потенциальных клиентов в покупателей. Этот процесс включает в себя анализ рынка и конкурентов, определение уникального ценностного предложения программы, разработку маркетинговых материалов (сайт-визитка, коммерческие предложения, презентации) и выбор каналов продвижения. Важным этапом является активное взаимодействие с потенциальными клиентами через демонстрации продукта, участие в специализированных выставках и конференциях, а также прямые продажи. Главная задача – донести до потенциальных пользователей (центры обработки данных, правоохранительные органы, разработчики, системные администраторы) преимущества программы по сравнению с существующими аналогами, особенно в части работы с большими бинарными файлами и множественного сравнения.

Внедрение и поддержка продукта охватывает процессы, следующие за продажей программного обеспечения. Внедрение включает в себя предоставление клиенту установочных файлов программы, а при необходимости – оказание помощи в её развертывании и первоначальной настройке в IT-инфраструктуре клиента. Последующая поддержка продукта предполагает оперативное решение возникающих технических проблем, предоставление обновлений, включающих исправления ошибок и новые функции, а также консультационную помощь

пользователям. Качественная и своевременная поддержка является ключевым фактором для формирования лояльности клиентов и поддержания долгосрочных отношений.

Управление проектом и бизнес-процессами обеспечивает общую координацию всех этапов стартап-проекта. Это включает в себя планирование ресурсов (человеческих, финансовых, технических), постановку задач, контроль выполнения, управление рисками и принятие стратегических решений. Важным аспектом является формирование эффективной команды, включающей разработчиков и специалистов по продажам и поддержке. Также в рамках этого процесса осуществляется юридическое оформление деятельности, включая регистрацию интеллектуальной собственности на разработанное программное обеспечение, что обеспечивает защиту прав на продукт.

3.3 Каналы взаимодействия

Для успешного продвижения и реализации программного продукта по побитовому сравнению файлов большого объема необходимо разработать эффективные каналы взаимодействия с потенциальными клиентами. Выбор правильных каналов позволит максимально охватить целевую аудиторию и донести ценность предлагаемого решения.

Одним из основных способов привлечения клиентов станет «Сайт-визитка» или специализированная целевая страница (landing page). Этот ресурс предоставит потенциальным заказчикам круглосуточный доступ к исчерпывающей информации о программе, её функциональных возможностях, преимуществах и сценариях применения. На сайте будет подробно описан весь предлагаемый функционал системы, включая работу с большими файлами, множественное сравнение и детализированную визуализацию отличий. Контактные формы и возможность обратной связи облегчат коммуникацию с будущими клиентами, позволяя им задавать вопросы и оставлять запросы на демонстрацию или приобретение продукта.

					ПГУ1.090301.10.001 ПЗ	Лист
						48
Изм.	Лист	№ докум.	Подпись	Дата		

Для более эффективной демонстрации функционала и ценности информационной системы необходимо, путем прямого обзвона и рассылки коммерческих предложений, согласовать личные встречи с руководителями IT-отделов, системными администраторами, специалистами по информационной безопасности и разработчиками в целевых организациях (центры обработки данных, правоохранительные органы, научно-исследовательские институты, крупные производственные предприятия). На таких встречах можно будет показать полный спектр возможностей продукта в действии, продемонстрировать его производительность на реальных данных клиента и ответить на специфические вопросы потенциальных пользователей.

После заключения сделки, команда будет отвечать за поставку и внедрение информационной системы. Это включает предоставление установочных пакетов, при необходимости – помощь в развертывании и настройке программы в соответствии с инфраструктурой и потребностями клиента. Четкое планирование и организация этапов внедрения помогут сэкономить время и обеспечить плавный переход к полноценному использованию системы. Также необходимо обеспечить поддержку клиентов в течение оговоренного периода после приобретения продукта. Это включает решение технических проблем, предоставление обновлений и доработок системы в соответствии с потребностями клиента, а также консультации и обучение пользователей по использованию программы. Быстрый и качественный ответ на запросы клиентов поможет создать положительное впечатление от сотрудничества и сформировать лояльную клиентскую базу. Чтобы поддерживать долгосрочные отношения с клиентами, нужно стремиться регулярно предоставлять обновления и новости о продукте через электронные рассылки. Также целесообразно организовывать периодические вебинары или онлайн-сессии для сбора обратной связи и обсуждения возможностей системы.

3.4 Источники дохода

Для оценки экономической целесообразности и перспектив стартап-проекта по разработке программного обеспечения для побитового сравнения файлов необходимо провести анализ потенциальных потоков доходов, определить ключевые аспекты взаимодействия с клиентами и выполнить предварительную оценку затрат и доходов.

Основным источником доходов будет прямая продажа разработанного программного обеспечения (лицензий на использование). Дополнительные потоки доходов могут включать:

- подписка на регулярные обновления и техническую поддержку;
- партнерские программы с дистрибьюторами и реселлерами, расширяющие охват рынка.

Для поддержания долгосрочных и продуктивных отношений с клиентами планируется:

- предоставление качественной технической поддержки и консультаций по вопросам использования программы и решения возникающих проблем;
- регулярное обновление программного обеспечения с учетом обратной связи от пользователей, что позволит улучшать продукт и поддерживать его актуальность.

Предварительная оценка затрат и доходов на момент выхода на самоокупаемость, с учетом масштабирования проекта, выглядит следующим образом:

Затраты:

- Заработная плата (с учетом отчислений) - 321.5 тыс. рублей;
- Сырье и материалы (лицензии, оборудование) - 157 тыс. рублей;
- Аренда помещений - 178.5 тыс. рублей;
- Коммунальные услуги - 36 тыс. рублей;

– Прочие затраты (маркетинг, реклама) - 143 тыс. рублей Итого расходы: 836 тыс. рублей.

Доходы: продажа программного обеспечения - 1286 тыс. рублей (приблизительно 429 лицензий).

Чистая прибыль: 1286 тыс. рублей - 836 тыс. рублей = 450 тыс. Рублей.

При указанных затратах и доходах, срок окупаемости проекта составит: Первоначальные затраты: 836 тыс. рублей Чистый ежемесячный доход: 450 тыс. рублей / 12 мес. = 37.5 тыс. рублей Срок окупаемости: 836 тыс. рублей / 37.5 тыс. рублей $\approx$ 22.29 месяца.

Таким образом, при достижении заявленных финансовых показателей, проект окупится менее чем за 2 года.

3.5 Ключевые ресурсы

Для успешного запуска, полноценного функционирования и дальнейшего развития стартап-проекта по разработке программного обеспечения для побитового сравнения файлов потребуются определенные ключевые ресурсы. Эффективное управление этими ресурсами является залогом успешной реализации проекта и достижения его бизнес-целей. В нашем случае это будут:

- разработчики программного обеспечения, специализирующиеся на системном программировании, c++ и windows api;
- менеджеры по продажам, обладающие навыками ведения переговоров с корпоративными клиентами и продвижения b2b-продуктов;
- компьютеры, серверы (для разработки и, возможно, для хостинга сайта-визитки), специализированное программное обеспечение (лицензии на ide, инструменты тестирования) и другое техническое оборудование, необходимое для разработки и обслуживания системы;
- офисное пространство для работы команды и проведения встреч с клиентами;

– коммуникационное оборудование для связи с клиентами, такое как телефоны и компьютеры с доступом в интернет;

– финансовые ресурсы для финансирования разработки, маркетинга, продаж и операционных расходов бизнеса.

Финансирование проекта планируется осуществить как за счет собственных средств инициатора проекта, так и за счет государственной поддержки, например, в рамках программы «Гранты для молодых предпринимателей».

3.6 Риски проекта

Реализация стартап-проекта по разработке программного обеспечения для побитового сравнения файлов может быть подвержена различным рискам и форс-мажорным обстоятельствам, которые способны оказать негативное влияние на его успех. Важно провести анализ и предусмотреть меры по минимизации возможных негативных последствий таких обстоятельств.

Возможными рисками являются, например:

– недостаточный спрос на предлагаемое программное обеспечение: несмотря на выявленную актуальность, рынок может оказаться менее восприимчивым к новому продукту, чем ожидалось, что может быть связано с недостаточной осведомленностью потенциальных клиентов о преимуществах побитового сравнения или сильной инерцией в использовании существующих, пусть и менее функциональных, инструментов;

– конкуренция со стороны уже существующих аналогов: хотя проведенный анализ показал недостатки текущих решений, крупные игроки рынка могут выпустить обновленные версии своих продуктов или новые инструменты, предлагающие схожий функционал, что усилит конкурентное давление;

– проблемы с финансированием проекта: несмотря на первоначальное планирование, могут возникнуть непредвиденные расходы или задержки с получением запланированных инвестиций (например, грантов), что негативно скажется на темпах разработки и маркетинговых активностях;

– изменение законодательства или стандартов: хотя программа является утилитой общего назначения, изменения в законодательстве, касающиеся хранения, обработки или верификации цифровых данных (например, в сфере информационной безопасности или судебной экспертизы), могут потребовать адаптации функционала программы;

– риски информационной безопасности: хотя сама программа является десктопным приложением, она работает с чувствительными данными пользователей. риски, связанные с возможными уязвимостями в коде, которые могут быть использованы для несанкционированного доступа к данным, или риски, связанные с целостностью обрабатываемых файлов, должны быть минимизированы;

– нестабильность экономической и политической ситуации: общие макроэкономические факторы, такие как инфляция, изменения курсов валют, или политические события, могут повлиять на покупательную способность целевой аудитории, стоимость разработки и общую инвестиционную привлекательность проекта.

Заключение

Результат выполнения данной выпускной квалификационной работы можно считать успешно реализованным в соответствии с заданием – специализированное программное обеспечение для высокопроизводительного и точного побитового сравнения файлов большого объема.

В ходе выполнения работы были успешно выполнены следующие задачи:

– анализ предметной области: были детально исследованы потребности пользователей и выявлены ограничения существующих решений в области побитового сравнения файлов, особенно при работе с большими объемами данных и необходимостью сравнения множества объектов. это позволило определить ключевые функциональные требования и архитектурные особенности будущей системы;

– выбор инструментов разработки: на основе проведенного анализа было принято решение о разработке программы на языке c и c++ с использованием microsoft visual studio 2019 и нативного windows api. данный выбор обоснован необходимостью достижения максимальной производительности, низкоуровневого контроля над системными ресурсами и эффективного управления памятью;

– анализ актуальности разработки: была подтверждена высокая актуальность создания такого программного продукта в условиях растущих объемов цифровых данных и требований к их целостности, а также отсутствия полноценных аналогов на рынке;

– анализ аналогов информационных систем: проведенный обзор существующих решений (total commander, winhex, diffmerge) выявил их ограничения по объему обрабатываемых файлов, количеству сравниваемых объектов и детализации побитовой визуализации, что подчеркнуло уникальность и востребованность разрабатываемой программы;

– анализ архитектуры и структура программы: была разработана модульная архитектура приложения, включающая компоненты для пользовательского интерфейса, файлового ввода-вывода, побитового сравнения и визуализации различий, что обеспечивает гибкость и масштабируемость системы;

– постановка задачи: четко сформулированы функциональные и нефункциональные требования к программе, включая возможность сравнения до 9 файлов, работу с memory-mapped files и «скользящим окном», а также наглядную визуализацию отличий;

– реализация функционала системы: были реализованы основные модули программы, включая высокоэффективный механизм чтения файлов с использованием memory-mapped files и поблочной обработкой данных, алгоритм побитового сравнения, а также интуитивно понятный графический интерфейс пользователя, построенный на чистом windows api. внедрена система цветового кодирования для наглядного отображения различий и функции навигации по ним.

В результате работы была создана программа, которая предоставляет пользователю мощный, быстрый и надежный инструмент для побитового сравнения файлов любого объема. Она позволяет эффективно верифицировать целостность данных, выявлять мельчайшие модификации и анализировать различия между множеством версий файлов, что является критически важным для специалистов в области IT, информационной безопасности, разработки программного обеспечения и криминалистики.

Перспективы развития системы включают расширение функционала (например, добавление возможности сохранения отчетов о сравнении, интеграции с системами контроля версий, поддержки сетевых путей), дальнейшую оптимизацию производительности для экстремально больших файлов и, возможно, адаптацию для других операционных систем. Разработанное программное обеспечение готово к использованию и может значительно повысить эффективность процессов контроля и анализа цифровых данных.

Список использованных источников

1. Рихтер, Джеффри. «Windows через C/C++: Программирование на уровне ядра», 2008. – 896с.
2. Петцольд, Чарльз. «Программирование для Windows», 2000. – 1300с.
3. Microsoft Learn. Документация по Windows API: [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/windows/win32/> (дата обращения 03.06.2025).
4. Таненбаум, Эндрю С., Бос, Герберт. «Современные операционные системы», 2015. – 1120с. (Для общих знаний об ОС и файловых системах).
5. Habr.com.: [Электронный ресурс]. URL: <https://habr.com/ru> (дата обращения 03.06.2025).

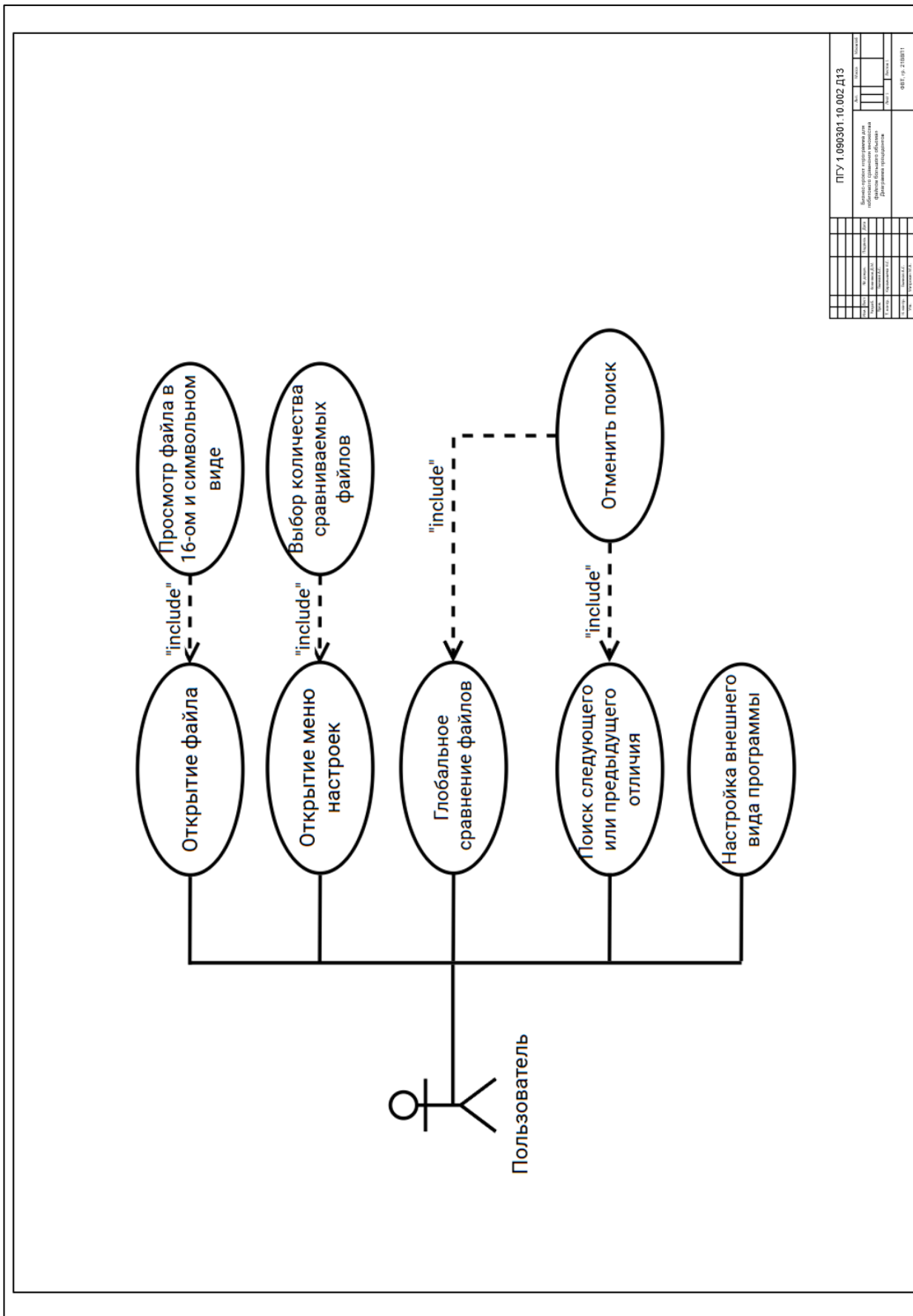
Приложение А

(Обязательное)

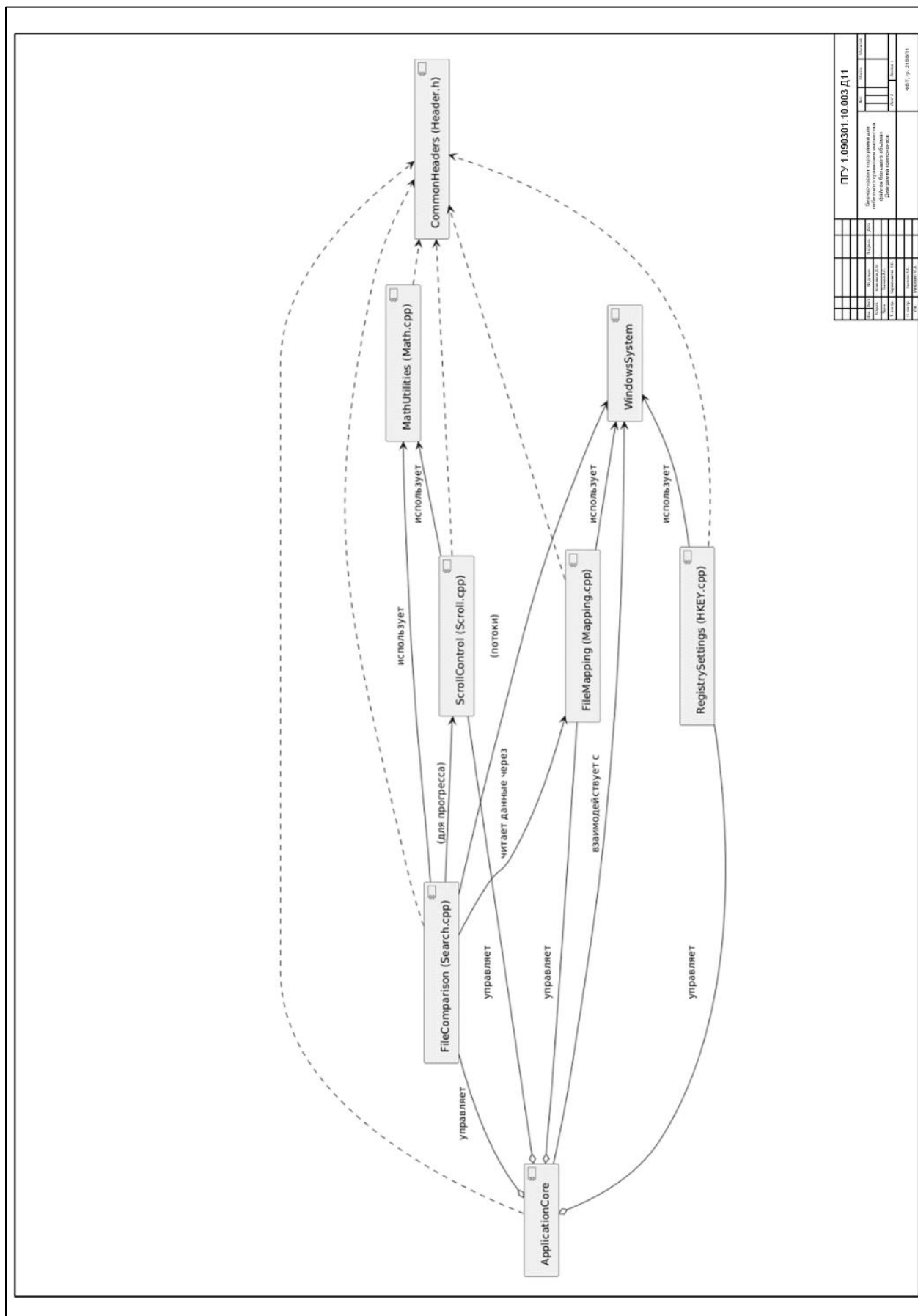
Бизнес-проект «Программа для побитового сравнения множества файлов
большого объема»
UML диаграммы

					ПГУ1.090301.10.001 ПЗ	Лист
						57
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 – UML – диаграмма прецедентов пользователя



Приложение А.2 – UML – диаграмма компонентов



ПГУ 1.090301.10.003 Д11									
№ докум.	№ докум.	№ докум.	№ докум.	№ докум.	№ докум.	№ докум.	№ докум.	№ докум.	№ докум.
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10

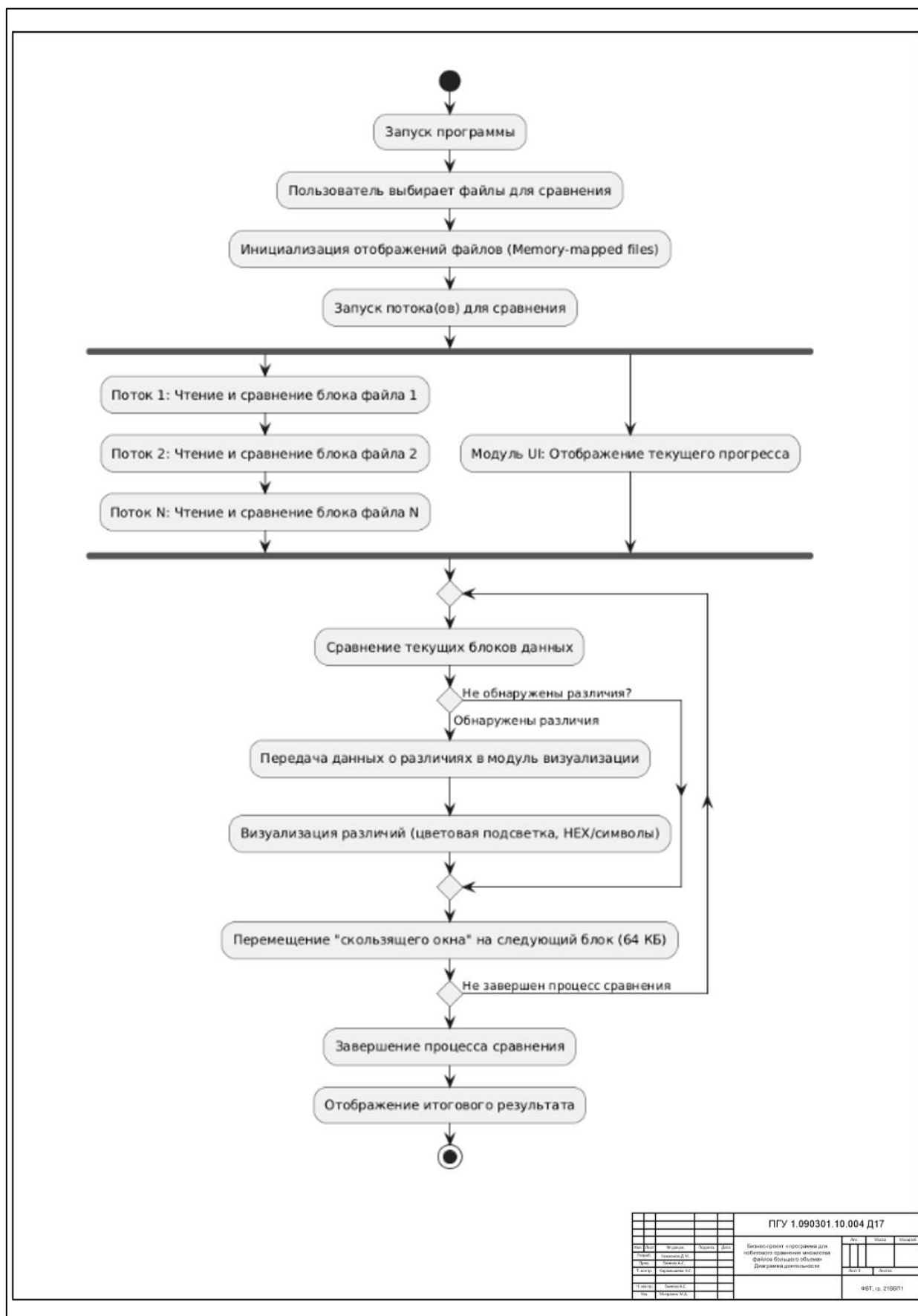
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.10.001 ПЗ

Лист

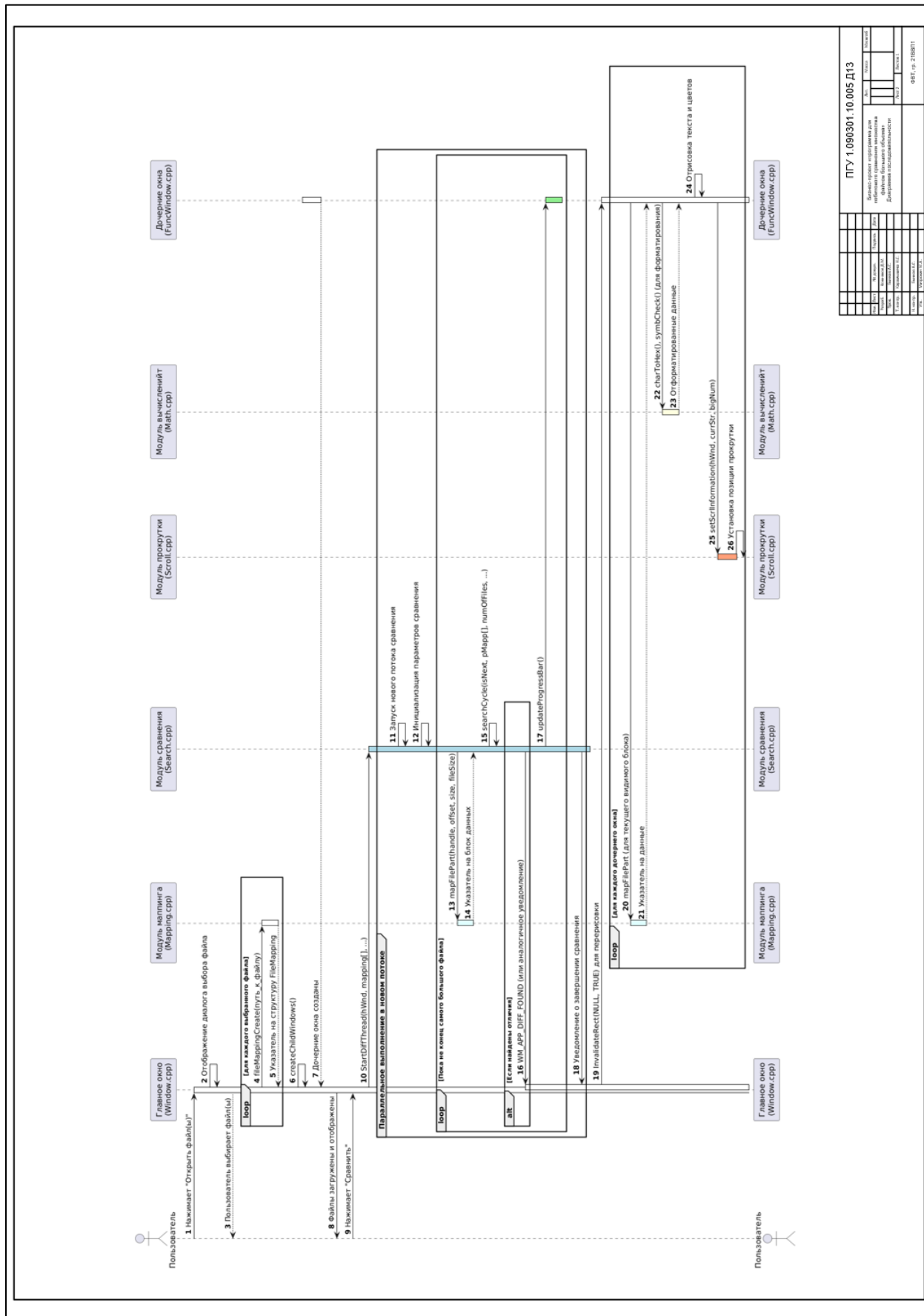
59

Приложение А.3 – UML – диаграмма деятельности



ПГУ 1.090301.10.004 Д17				
Изм.	Лист	№ докум.	Подпись	Дата
1	1			
2	2			
3	3			
4	4			
5	5			
6	6			
7	7			
8	8			
9	9			
10	10			

Приложение А.4 – UML – диаграмма последовательности



Приложение Б

(Обязательное)

Бизнес-проект «Программа для побитового сравнения множества файлов
большого объема»

Листинг программы

					ПГУ1.090301.10.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		62

Файл «main.cpp»

```
#define _CRT_SECURE_NO_WARNINGS

#include <windows.h>
#include "Header.h"
#include <commdlg.h>

extern HWND hButtonOpenSettings{}, hButtonNext{}, hButtonPrev{},
hComboBoxFont{},
hComboBoxSize{}, hProgressBar{}, hProgressBarForCheck{},
hSettingsWnd{}, hSettingsButton{},
hNumOfFilesComboBox{}, hInfoButton{}, hChangeTheme{}, hText{},
hCompare{};
extern HWND hChildDiff[MAXNUMOFFILES] {},
hButtonFileCreate[MAXNUMOFFILES] {},
hHorzScrollBar[MAXNUMOFFILES] {},
hBoxNameOfFile[MAXNUMOFFILES] {}, hTooltip[MAXNUMOFFILES] {};

// массив цветов для различных файлов
extern COLORREF gFileColors[MAXNUMOFFILES + 1] = { RGB(255,0,0),
RGB(255,255,255), RGB(130,230,30),
RGB(150,50,250) ,RGB(210,250,50) ,RGB(250,50,250)
,RGB(40,130,40),
RGB(50,150,250),RGB(50,200,50),RGB(50,250,250) };

extern BOOL SearchIsON = FALSE;
extern TOOLINFO ti[MAXNUMOFFILES] = { 0 };
extern COLORREF gThemeColors[4] = {
RGB(255,255,255),RGB(20,20,20) , RGB(35,35,35), RGB(0,0,0) };

// Точка входа в приложение
int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE
hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {
    // Регистрация класса главного окна
    static TCHAR szMainClass[] = _T("MainDiffer");
    WNDCLASS wcMain = { 0 };
    wcMain.lpfnWndProc = WindowProc; // Функция обработки
сообщений
    wcMain.hInstance = hInstance;
    wcMain.lpszClassName = szMainClass;

    if (!RegisterClass(&wcMain)) {
        MessageBox(NULL, REGISTRATIONFAILEDMAIN, L"Ошибка",
MB_ICONERROR);
        return 0;
    }

    // Регистрация класса дочернего окна
    static TCHAR szChildClass[] = _T("ChildDiffer");
    WNDCLASS wcChild = { 0 };
```

```

        wcChild.lpfWndProc = ChildWindowProc; // Функция обработки
сообщений для дочерних окон
        wcChild.hInstance = hInstance;
        wcChild.lpszClassName = szChildClass;

        if (!RegisterClass(&wcChild)) {
            MessageBox(NULL, REGISTRATIONFAILEDCHILD, L"Ошибка",
MB_ICONERROR);
            return 0;
        }

        // Регистрация класса окна настроек
        static TCHAR szSettingsClass[] = _T("Settings");
        WNDCLASS wcSettings = { 0 };
        wcSettings.lpfWndProc = SettingsWindowProc; // Функция
обработки сообщений для окна настроек
        wcSettings.hInstance = hInstance;
        wcSettings.lpszClassName = szSettingsClass;

        if (!RegisterClass(&wcSettings)) {
            MessageBox(NULL, REGISTRATIONFAILEDSETTINGS, L"Ошибка",
MB_ICONERROR);
            return 0;
        }

        // Создание главного окна
        HWND hWnd = CreateWindowEx(
            WS_EX_OVERLAPPEDWINDOW, szMainClass, L"Программа для
сравнения файлов Diff",
            WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT,
            CW_USEDEFAULT, CW_USEDEFAULT,
            NULL, NULL, hInstance, NULL);

        if (!hWnd) {
            MessageBox(NULL, REGISTRATIONFAILEDWINDOW, L"Ошибка",
MB_ICONERROR);
            return 0;
        }

        ShowWindow(hWnd, nCmdShow);
        UpdateWindow(hWnd);

        // Основной цикл обработки сообщений
        MSG msg;
        while (GetMessage(&msg, NULL, 0, 0)) {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }

        return (int)msg.wParam;
    }

```

Файл «header.h»

```
#pragma once
#include <iostream>
#include <windows.h>
#include <tchar.h>
#include <cmath>
#include <commdlg.h>
#include <CommCtrl.h>

#pragma comment(lib, "comctl32.lib")
#pragma comment(linker, "/manifestdependency:\"type='win32'
name='Microsoft.Windows.Common-Controls' version='6.0.0.0'
processorArchitecture='*' publicKeyToken='6595b64144ccf1df'
language='*'\")

#pragma region Consts

//-----
// Определение констант для элементов интерфейса
//-----

#define COMBOBOX_FONT 1
#define COMBOBOX_SIZE 2
#define BUTTON_OPENSETTINGS 3
#define BUTTON_NEXT 4
#define BUTTON_PREV 5
#define BUTTON_FILEOPEN 6
#define SETTINGS_WINDOW 7
#define SETTINGS_BUTTON 8
#define SETTINGS_COMBOBOX 9
#define HORIZONTAL_SCROLL 10
#define BUTTON_CLOSESETTINGS 11
#define BUTTON_INFO 12
#define BUTTON_CANCELSETTINGS 13
#define BUTTON_CHANGETHHEME 14
#define BUTTON_COMPARE 15

#define LINELENGTH 16 // количество символов на одну
строку
#define MAXNUMOFFILES 11 // максимальное количество
файлов

//-----
// сообщения для MessageBox
//-----

#define BUTTONINFOTEXT L"Программа Diff позволяет производить
сравнение от 2 до 9 файлов одновременно.\n Изменение кол - во
файлов производится через меню настроек.\n Для нахождения
```

```

следующего/предыдущего отличия нажать кнопки (>>>)/(<<<)
соответственно.\n Для прерывания поиска нажать кнопку Esc"
#define STOPSEARCH L"Прекратить поиск?"
#define STOPSEARCHWINDOW L"Заккрытие окна прекратит поиск.
Продолжить?"
#define STOPSEARCHOPENFILE L"Изменение файла прекратит поиск.
Продолжить?"
#define ERRCREATEHKEY L"Произошла ошибка при создании ключа"
#define NODIFFERENCE L"Нет различий"
#define NODIFFERENCEINFILES L"В этих файлах нет различий"
#define HAVEDIFFERENCEINFILES L"В этих файлах есть различия"
#define REGISTRATIONFAILEDMAIN L"Регистрация главного окна не
удалась!"
#define REGISTRATIONFAILEDCHILD L"Регистрация дочернего окна не
удалась!"
#define REGISTRATIONFAILEDSETTINGS L"Регистрация окна настроек
не удалась!"
#define REGISTRATIONFAILEDWINDOW L"Создание окна не удалось!"

#pragma endregion

extern COLORREF gFileColors[MAXNUMOFFILES + 1];
extern COLORREF gThemeColors[4];

//-----
// Прототипы функций обратного вызова для окон
//-----
LRESULT CALLBACK WindowProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK ChildWindowProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK SettingsWindowProc(HWND, UINT, WPARAM, LPARAM);

#pragma region Structs

//-----
// структура для передачи данных в функции вывода текста на
// экран
//-----
struct ForTextOut {
    HDC hdc;
    size_t currStr;
    int x;
    int fontSize;
    int currFile;
    int numOfFiles;
    int numOfFilesOpened;
};

```

```

struct ForTextFormingCycle {
    int numOfFiles;
    int u;
    int k;
    int currFile;
    size_t endFileNum;
};

//-----
// Структура для хранения информации о файловом отображении в
// памяти
//-----

struct FileMapping {
    HANDLE hFile;        // Дескриптор файла
    HANDLE hMapping;     // Дескриптор отображения файла
    size_t fsize;        // Размер файла
};

//-----
// Структура для передачи hwnd кнопок
//-----

struct Buttons {
    HWND Butn1;
    HWND Butn2;
    HWND Butn3;
    HWND ButtnSearch;
    HWND hProgressBar;
    HWND hCompare;
};

//-----
// Структура для хранения данных и указателей на отображенные
// данные
//-----

struct DataPoint {
    char* data;           // Буфер данных
    size_t lastPoint;     // Конечная позиция данных
    size_t fsize;         // Размер файла
    int currLocation;     // Текущая позиция
    unsigned char* mappedData; // Указатель на данные,
    возвращенные MapViewOfFile

    DataPoint() : data(new char[128 * 1024]), lastPoint(0),
    fsize(0), currLocation(0), mappedData(nullptr) {}

```

```

        DataPoint(size_t dataSize) : data(new char[dataSize]),
lastPoint(0), currLocation(0), fsize(0), mappedData(nullptr) {
        std::fill(data, data + dataSize, 0); // Заполнение
нулями
    }

    ~DataPoint() {
        if (data) {
            delete[] data; // Освобождение памяти буфера
        }
    }
};

//-----
//-----
// Параметры потока для многопоточного выполнения
//-----
//-----

struct ThreadParams {
    BOOL checking = FALSE; // Флаг проверки
    int k; // Дополнительный параметр
    int result; // Результат работы потока
    BOOL isNext; // Указывает направление поиска
    HWND hWnd; // Дескриптор окна
    FileMapping** threadMapping; // Массив отображений файлов,
используемых в потоке
    HWND hWndButton1; // Дескриптор кнопки 1
    HWND hWndButton2; // Дескриптор кнопки 2
    HWND hWndButton3; // Дескриптор кнопки 3
    HWND hProgressBar; // Дескриптор индикатора выполнения
    HWND BtnnSearch;
    HWND hCompare;
    int numFiles; // Количество файлов

    ~ThreadParams() {}
};

#pragma endregion

#pragma region Mapping

/// <summary>
/// Функция для отображения части файла в памяти с заданным
смещением и размером
/// </summary>
/// <param name="hMapping"></param>
/// <param name="offset">Размер одного буфера</param>
/// <param name="size">Размер уже обработанного файла</param>
/// <param name="fileSize">Размер файла</param>
/// <returns>Возвращает указатель на данные маппинга</returns>

```

```

unsigned char* mapFilePart(HANDLE hMapping, uint64_t offset,
size_t size, size_t fileSize);

/// <summary>
/// Точка до которой мы может "мапить" данные
/// </summary>
/// <param name="size">Размер уже обработанного файла</param>
/// <param name="offset">Размер одного буфера</param>
/// <param name="fileSize">Размер файла</param>
/// <returns>Возвращает скорректированный размер</returns>
size_t deathPoint(uint64_t offset, size_t size, size_t
fileSize);

/// <summary>
/// Создание отображения файла
/// </summary>
/// <param name="fname">Имя файла</param>
/// <returns>Возвращает указатель на маппинг файла</returns>
FileMapping* fileMappingCreate(const wchar_t* fname);

//-----
// Закрытие отображения файла и освобождение ресурсов
//-----

void fileMappingClose(FileMapping* mapping);

#pragma endregion

#pragma region Math

/// <summary>
/// Нахождение максимального размера файла, не может быть
вызвана если никакой файл не открыт
/// </summary>
/// <param name="pMapp">Указатель на структуру с маппингом
файлов</param>
/// <param name="numOfFiles">Количество файлов</param>
/// <returns>Возвращает размер самого большого файла, если не
открыт ни один файл вернет 0</returns>
size_t bigger(FileMapping* mapp[], int num);

/// <summary>
/// Проверка на нахождение числа в промежутке
/// </summary>
/// <param name="a">Нижняя граница</param>
/// <param name="b">Верхняя граница</param>
/// <param name="value">Проверяемое число</param>
/// <returns>Возвращает число если входит в промежуток</returns>

```

```

int between(int a, int b, int value);

//-----
//-----
// Перевод числа в 16-ую СС
//-----
//-----

char digitToHex(int digit);

/// <summary>
/// Перевод char символа в 16-ую СС
/// </summary>
/// <param name="c">Изначальный символ char</param>
/// <param name="str">Строка куда будет записан
результат</param>
void charToHex(byte c, char str[2]);

/// <summary>
/// Проверка на наличие системных символов
/// </summary>
/// <param name="strTemp">Строка, котрая будет выводится на
экран</param>
/// <param name="member">Структура с данными файлов</param>
/// <param name="u">Номер символа в строке</param>
/// <param name="k">Номер символа в прочитанном буфере</param>
void symbCheck(unsigned char strTemp[32], DataPoint& member, int
u, int k);

/// <summary>
/// Перевод числа в 16 СС
/// </summary>
/// <param name="strTempJohn">Строка, котрая будет выводится на
экран</param>
/// <param name="member">Структура с данными файлов</param>
/// <param name="u">Номер символа в строке</param>
/// <param name="k">Номер символа в прочитанном буфере</param>
/// <param name="strJohn">???? Строка для временного хранения
символов  ?????</param>
void symbCheckSec(char strTempJohn[48], DataPoint& member, int
u, int k, char strJohn[2]);

/// <summary>
/// Заполнения строки 3-мя пробелами
/// </summary>
/// <param name="strTempJohn">Строка куда заполняются
пропуски</param>

```

```

/// <param name="u">Место в строке откуда, включительно, будет
заполнено 3 пробела</param>
void symbCheckThird(char strTempJohn[48], int u);

#pragma endregion

#pragma region Scroll

/// <summary>
/// Проверка на необходимость горизонтального скролла
/// </summary>
/// <param name="hWnd">Окно где нужно произвести
проверку</param>
/// <param name="fontSize">Размер шрифта</param>
/// <returns>Возвращает TRUE если горизонтальный скролл
нужен</returns>
BOOL doNeedHorz(HWND hWnd, int fontSize);

/// <summary>
/// Проверка на необходимость вертикального скролла
/// </summary>
/// <param name="greater">Размер большего из всех файлов</param>
/// <param name="size">Количество строчек, которое поместится на
экран</param>
/// <returns>Возвращает TRUE если вертикальный скролл
нужен</returns>
BOOL doNeedVert(size_t greater, int size);

/// <summary>
/// Обновление вертикального скролла для всех окон
/// </summary>
/// <param name="hWnd">Окно для которого нужно установить
позицию вертикального скролла</param>
/// <param name="currStr">Текущая строка</param>
/// <param name="num">Размер большего из всех файлов</param>
void setScrlInformation(HWND hWnd, double currStr, double num);

/// <summary>
/// Расчет нужного положения вертикального скролла
/// </summary>
/// <param name="cur">Текущее положение скролл бара</param>
/// <param name="max">Максимальное положение скролл бара</param>
/// <param name="pMapping">Указатель на структуру с маппингом
файлов</param>
/// <param name="numOfFiles">Количество файлов</param>
/// <param name="sizeWindow">Количество строчек, которое
поместится на экран</param>
/// <returns>Возвращает номер строки соответствующий положению
скролл бара</returns>

```

```

size_t vertScroll(float cur, float max, FileMapping* mapping[],
int numFiles, int sizeWindow);

#pragma endregion

#pragma region FuncWindow
/// <summary>
/// Функция для удаления окон файлов и кнопок
/// </summary>
/// <param name="numFiles">Количество файлов</param>
/// <param name="hChildDiff">Указатель на окно отрисовки</param>
/// <param name="hButton"></param>
/// <param name="hWnd"></param>
/// <returns></returns>
int changeNumFiles(int numFiles, HWND hChildDiff[], HWND
hButton[], HWND hWnd);

//-----
// Добавление шрифтов в комбобокс для выбора
//-----
void addFontCombox(HWND hCombox);

//-----
// Добавление размеров шрифтов в комбобокс
//-----
void addSizeCombox(HWND hCombox);

//-----
// Добавление количества файлов в комбобокс
//-----
void addFilesCombox(HWND hCombox);

/// <summary>
/// Обновляет заполнение прогресс бара
/// </summary>
/// <param name="hProgressBar">Указатель на окно Прогресс
бара</param>
/// <param name="k">Номер символа в прочитанном буфере</param>
/// <param name="placeInFile">Номер буфера в файле</param>
/// <param name="bigger">Размер большего из всех файлов</param>

```

```

void updateBar(HWND hProgressBar, double k, int placeInFile,
double bigger);

//-----
// Расчет сколько строчек может поместиться на экране
//-----
int whatSize(HWND hWnd, int hieght);

//-----
// Обновление окон (отрисовка)
//-----
void refresh(int numOfFiles, HWND hChildDiff[]);

/// <summary>
/// Обновление начальной строки
/// </summary>
/// <param name="numOfFiles">Количество файлов</param>
/// <param name="hChildDiff">Указатель на окно отрисовки</param>
/// <param name="currStr">Номер текущей строки</param>
void updateStr(int numOfFiles, HWND hChildDiff[], size_t
currStr);

#pragma endregion

#pragma region For file printing

/// <summary>
/// Формирует пустую строку
/// </summary>
/// <param name="strForHex">Строка с 16-ым отображением</param>
/// <param name="strForData">Строка с обычным
отображением</param>
/// <param name="u">Номер символа в строке</param>
/// <param name="numOfFiles">Количество файлов</param>
void stringBlanker(char strForHex[9][50], char
strForData[9][18], int u, int numOfFiles);

/// <summary>
/// Печатает адрес выводимых символов
/// </summary>
/// <param name="hdc">hdc экрана</param>
/// <param name="currStr">Номер текущей строки</param>
/// <param name="strLength">Количество символов из файла на одну
строку</param>

```

```

/// <param name="x">Координата x</param>
/// <param name="y">Координата y</param>
/// <returns></returns>
size_t adrPrinting(HDC hdc, size_t currStr, int strLength, int
x, int y);

/// <summary>
/// Формирует строку нужного цвета
/// </summary>
/// <param name="strForHex">Строка с 16-ым отображением</param>
/// <param name="strForData">Строка с обычным
отображением</param>
/// <param name="data">Структура с данными файлов</param>
/// <param name="fileNum">Номер файла</param>
/// <param name="currFile">Номер текущего файла</param>
/// <param name="u">Номер символа в строке</param>
/// <param name="k">Номер символа в прочитанном буфере</param>
void stringMaker(char strForHex[9][50], char strForData[9][18],
DataPoint* data, int num, int currFile, int u, int k);

/// <summary>
/// Отрисовывает строки на экран в соответствии с их цветами
/// </summary>
/// <param name="strForHex">Строка с 16-ым отображением</param>
/// <param name="strForData">Строка с обычным
отображением</param>
/// <param name="y">Координата y</param>
/// <param name="txtData">Структура с основными данными для
отрисовки строчек</param>
void stringPrint(char strForHex[9][50], char strForData[9][18],
int y, ForTextOut txtData);

/// <summary>
/// Сравнивает символы всех открытых файлов
/// </summary>
/// <param name="strForHex">Строка с 16-ым отображением</param>
/// <param name="strForData">Строка с обычным
отображением</param>
/// <param name="cycleData">Структура с данными основного цикла
сравнения файлов</param>
/// <param name="data">Структура с данными файлов</param>
void mainCycle(char strForHex[9][50], char strForData[9][18],
ForTextFormingCycle cycleData, DataPoint* pData);

//-----
// Создание буфера 128 кбайт для просмотра данных

```

```

//-----
void buffMaker(FileMapping* mapping, int currPosInFile,
DataPoint& member);

//-----
// Основная функция отрисовки и сравнения файлов
//-----

void diffTxtOut(int numOfShownedStr, int k, DataPoint* data,
size_t Greater, ForTextOut tOut);

/// <summary>
/// Функция заполнения буфера (при необходимости)
/// </summary>
/// <param name="pMapping">Указатель на структуру с маппингом
файлов</param>
/// <param name="hWnd">Указатель на окно отрисовки</param>
/// <param name="tOut">Структура с переменными для вывода текста
на экран</param>
/// <returns></returns>
int txtOut(FileMapping* mapping[], HWND hWnd, ForTextOut tOut);

#pragma endregion

#pragma region Thread for diff
//-----
// Проверяет идет ли поиск
//-----

BOOL isSearching();

/// <summary>
/// Основной цикл поиска отличий
/// </summary>
/// <param name="isNext">В какую сторону идет поиска true-вниз,
false-вверх</param>
/// <param name="pMap">Структура маппинга файлов</param>
/// <param name="numOfFiles">Количество файлов</param>
/// <param name="placeInFile">Начальный сектор файла где ищем
отличие (64 Кбайт - один сектор)</param>
/// <param name="data">Структура хранящая значение
файлов</param>
/// <param name="k">Номер символа сравниваемого в данном
секторе</param>
/// <param name="hProgressBar">Указатель на окно прогресс
бара</param>

```

```

/// <returns>Возвращает либо номер ошибки, либо номер строки с
отличием, нужно обрабатывать получаемое значение</returns>
signed long long int searchCycle(bool isNext, FileMapping *
pMapp[], int numOfFiles, int placeInFile, DataPoint data[], int
k, HWND hProgressBar);

```

```

/// <summary>
/// Функция поиска отличий
/// </summary>
/// <param name="pMapp">Структура маппинга файлов</param>
/// <param name="k">Номер символа в файле</param>
/// <param name="isNext">В какую сторону идет поиска true-вниз,
false-вверх</param>
/// <param name="numOfFiles"> Количество файлов</param>
/// <param name="hProgressBar">Указатель на окно прогресс
бара</param>
/// <param name="checking">Общая проверка на отличия(TRUE) или
нахождение следующего отличия(FALSE) </param>
/// <returns>Возвращает либо номер ошибки, либо номер строки с
отличием, нужно обрабатывать получаемое значение</returns>
signed long long int searchForDiff(FileMapping * mapp[], size_t
k, bool isNext, int numOfFiles, HWND hProgressBar, BOOL
checking);

```

```

//-----
// Функция для работы нити на поиск
//-----
DWORD WINAPI ThreadProc(LPVOID lpParam);

```

```

/// <summary>
/// Создание и запуск нити поиска по кнопке
/// </summary>
/// <param name="hWnd">Указатель родительского окна</param>
/// <param name="isNext">В какую сторону идет поиска true-вниз,
false-вверх</param>
/// <param name="mapping">Указатель на структуру с маппингом
файлов</param>
/// <param name="currStr">Номер текущей строки</param>
/// <param name="btn">Структура с кнопками, которые нужно
отключить на время поиска</param>
/// <param name="numOfFiles">Количество файлов</param>
/// <param name="checking">Общая проверка на отличия(TRUE) или
нахождение следующего отличия(FALSE) </param>
void StartDiffThread(HWND hWnd, bool isNext, FileMapping *
mapping[], size_t currStr, Buttons btn, int numOfFiles);

```

```

//-----
// Создание и запуск нити сравнения файлов
//-----

void StartDiffThread(HWND hWnd, bool isNext, FileMapping *
mapping[], size_t currStr, Buttons btnn, int numFiles, BOOL
checking);

#pragma endregion

#pragma region HKEY

//-----
// Считывает значение из ключа char
//-----

TCHAR readHkey(_TCHAR szKeyName[], _TCHAR szBuf[]);

//-----
// Считывает значение из ключа int
//-----

int readIntHkey(_TCHAR szKeyName[], int szBuf[]);

//-----
// Создает и заполняет нужным значением ключ char
//-----

int createHkey(_TCHAR szKeyName[], _TCHAR szValue[], size_t
sizeofValue);

//-----
// Создает и заполняет нужным значением ключ int
//-----

int createIntHkey(_TCHAR szKeyName[], int szValue[], size_t
sizeofValue);
#pragma endregion

#pragma region WndProcCaseFunctions

/// <summary>
/// Меняет цвета темы и текста, а также записывает текущий цвет
в реестр

```

					ПГУ1.090301.10.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		77

```

/// </summary>
/// <param name="hWnd">указатель на главное окно</param>
/// <param name="themeName">Текущая тема</param>
/// <param name="THEME">Цвет основного окна</param>
/// <param name="UPPERTHEME">Цвет дополнительных окон</param>
void changeTheme(HWND hWnd, _TCHAR themeName[], COLORREF *THEME,
COLORREF *UPPERTHEME);

/// <summary>
/// Меняет размер шрифта и записывает его в реестр
/// </summary>
/// <param name="hComboBoxSize">Комбобокс размеров
шрифта</param>
/// <param name="fontSize">Переменная хранящая размер
шрифта</param>
/// <param name="wParam">WPARAM из функции окна</param>
/// <returns>Возвращает текущий размер шрифта</returns>
int changeSize(HWND hComboBoxSize, int fontSize, WPARAM wParam);

/// <summary>
/// Меняет шрифт и записывает текущий в реестр
/// </summary>
/// <param name="hComboBoxFont">Комбобокс списка шрифтов</param>
/// <param name="fontName">текущий шрифт</param>
/// <param name="wParam">WPARAM из функции окна</param>
/// <param name="size">Размер fontName</param>
void changeFont(HWND hComboBoxFont, _TCHAR fontName[30], WPARAM
wParam, size_t size);

/// <summary>
/// Рисует родительское окно
/// </summary>
/// <param name="hWnd">Указатель на родительское окно</param>
/// <param name="UPPERTHEME">Цвет родительского окна</param>
/// <param name="ps">Структура рисования</param>
/// <returns>Возвращает контекст отображения</returns>
HDC paintCase(HWND hWnd, COLORREF UPPERTHEME, PAINTSTRUCT ps);

/// <summary>
/// Создает все основные окна и элементы программы
/// </summary>
/// <param name="hWnd">Указатель на родительское окно</param>
void createMainWindows(HWND hWnd);

/// <summary>
/// Создает все дочерние элементы интерфейса и отображения
/// </summary>
/// <param name="hWnd">Указатель на родительское окно</param>
/// <param name="numOfFiles">Количество файлов</param>
void createChildWindows(HWND hWnd, int numOfFiles);

/// <summary>

```

```

/// Меняет размер объектов и окон при изменении размера
основного окна
/// </summary>
/// <param name="hWnd">Указатель на родительское окно</param>
/// <param name="numOfFiles">Количество файлов</param>
/// <param name="bigNum">Самый большой размер файла,
переведенный в строки</param>
/// <param name="sCurrStr">Номер текущей строки</param>
/// <param name="sFontSize">Размер шрифта</param>
void sizeChangedCase(HWND hWnd, int numOfFiles, size_t bigNum,
size_t sCurrStr, int sFontSize);

/// <summary>
/// Обрабатывает нажатия клавиш
/// </summary>
/// <param name="wParam">Параметры окна</param>
/// <param name="ScrollIsON">Флаг, включен ли скролл</param>
/// <param name="hWnd">Указатель на родительское окно</param>
/// <param name="numOfFiles">Количество файлов</param>
/// <param name="bigNum">Самый большой размер файла,
переведенный в строки</param>
/// <param name="sCurrStr">Номер текущей строки</param>
/// <param name="sFontSize">Размер шрифта</param>
size_t keydownCase(WPARAM wParam, BOOL ScrollIsON, HWND hWnd,
int numOfFiles, size_t bigNum, size_t sCurrStr, int sFontSize);
#pragma endregion

```

Файл «mapping.cpp»

```

#include "Header.h"
#pragma region Mapping

/// <summary>
/// Функция для отображения части файла в памяти с заданным
смещением и размером
/// </summary>
/// <param name="hMapping"></param>
/// <param name="offset">Размер одного буфера</param>
/// <param name="size">Размер уже обработанного файла</param>
/// <param name="fileSize">Размер файла</param>
/// <returns>Возвращает указатель на данные маппинга</returns>
unsigned char* mapFilePart(HANDLE hMapping, uint64_t offset,
size_t size, size_t fileSize) {
    if (offset >= fileSize) {
        return nullptr; // Если смещение за пределами файла,
возвращаем nullptr
    }

    // Корректируем размер, если оставшаяся часть файла меньше
запрашиваемого размера

```

```

        size_t mappingSize = (offset + size > fileSize) ? fileSize -
offset : size;

        DWORD high = static_cast<DWORD>((offset >> 32) &
0xFFFFFFFFFull); // Старшие 32 бита смещения
        DWORD low = static_cast<DWORD>(offset & 0xFFFFFFFFFull);
// Младшие 32 бита смещения

        unsigned char* dataPtr = (unsigned
char*)MapViewOfFile(hMapping, FILE_MAP_READ, high, low,
mappingSize);
        if (dataPtr == nullptr) {
            std::cerr << "mapFilePart - MapViewOfFile failed,
offset = " << offset << std::endl;
            return nullptr;
        }

        return dataPtr; // Возвращаем указатель на данные
    }

    /// <summary>
    /// Точка до которой мы может "мапить" данные
    /// </summary>
    /// <param name="size">Размер уже обработанного файла</param>
    /// <param name="offset">Размер одного буфера</param>
    /// <param name="fileSize">Размер файла</param>
    /// <returns>Возвращает скорректированный размер</returns>
    size_t deathPoint(uint64_t size, size_t offset, size_t fileSize)
    {
        size_t mappingSize = (size + offset > fileSize) ? fileSize -
size : offset;
        return mappingSize; // Возвращаем скорректированный размер
    }

    /// <summary>
    /// Создание отображения файла, после использования маппинга его
нужно закрыть
    /// </summary>
    /// <param name="fname">Имя файла</param>
    /// <returns>Возвращает указатель на маппинг файла</returns>
    FileMapping* fileMappingCreate(const wchar_t* pFname) {

        HANDLE hFile = CreateFile(pFname, GENERIC_READ,
FILE_SHARE_READ, nullptr, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL,
nullptr);
        if (hFile == INVALID_HANDLE_VALUE) {
            std::cerr << "fileMappingCreate - CreateFile failed,
fname = " << pFname << std::endl;
            return nullptr;
        }
    }

```

```

        LARGE_INTEGER dwFileSize;
        if (!GetFileSizeEx(hFile, &dwFileSize)) {
            std::cerr << "fileMappingCreate - GetFileSize failed,
fname = " << pFname << std::endl;
            CloseHandle(hFile);
            return nullptr;
        }

        HANDLE hMapping = CreateFileMapping(hFile, nullptr,
PAGE_READONLY, 0, 0, nullptr);
        if (hMapping == nullptr) {
            std::cerr << "fileMappingCreate - CreateFileMapping
failed, fname = " << pFname << std::endl;
            CloseHandle(hFile);
            return nullptr;
        }

        FileMapping* mapping =
(FileMapping*)malloc(sizeof(FileMapping)); // Выделение памяти
для структуры
        if (mapping == nullptr) {
            std::cerr << "fileMappingCreate - malloc failed, fname
= " << pFname << std::endl;
            CloseHandle(hMapping);
            CloseHandle(hFile);
            return nullptr;
        }

        // Инициализация структуры FileMapping
        mapping->hFile = hFile;
        mapping->hMapping = hMapping;
        mapping->fsize = static_cast<size_t>(dwFileSize.QuadPart);

        return mapping; // Возвращаем указатель на структуру
отображения файла
    }

    // Закрытие отображения файла и освобождение ресурсов
    void fileMappingClose(FileMapping* pMapping) {
        if (pMapping) {
            if (pMapping->hMapping) CloseHandle(pMapping->
hMapping); // Закрываем отображение
            if (pMapping->hFile) CloseHandle(pMapping->hFile);
            // Закрываем файл
            free(pMapping); // Освобождение выделенной памяти для
структуры
        }
    }

#pragma endregion

```

Файл «window.cpp»

```
#define _CRT_SECURE_NO_WARNINGS
#include "Header.h"

//-----
// Объявление глобальных переменных для окон и элементов
// управления
//-----

extern BOOL SearchIsON;
extern HWND hButtonOpenSettings, hButtonNext, hButtonPrev,
hComboBoxFont,
hComboBoxSize, hProgressBarr, hProgressBarForCheck,
hSettingsWnd, hSettingsButton,
hNumOfFilesComboBox, hInfoButton, hChangeTheme, hText, hCompare;
extern HWND hChildDiff[MAXNUMOFFILES],
hButtonFileCreate[MAXNUMOFFILES], hHorzScrollBar[MAXNUMOFFILES],
hBoxNameOfFile[MAXNUMOFFILES], hTooltip[MAXNUMOFFILES];
FileMapping* mapping[MAXNUMOFFILES]{};
extern TOOLINFO ti[MAXNUMOFFILES];
_TCHAR fontName[30] = L"Consolas";
_TCHAR themeName[10] = L"BLACK";
static int sFontSize = 10;
BOOL ScrollIsON = FALSE;
int numOffFiles = 2, numOffFilesOpened = 0;
COLORREF THEME = gThemeColors[1], UPPERTHEME =
gThemeColors[2];

//-----
// Считывает значения настроек из регистра шрифта
//-----

void hCheckFont(_TCHAR szKeyFont[]) {
    BOOL doesHaveIT = FALSE;
    if (readHkey(szKeyFont, fontName) == 65535) { // функция
        возвращает -1
        if (createHkey(szKeyFont, fontName, sizeof(fontName)) !=
0) {
            MessageBox(NULL, ERRCREATEHKEY, L"HKEY Error",
NULL);
        }
    }
    else {
        TCHAR szFontName[20]{};
        for (int i = 0; i < 3; i++) {
            SendMessage(hComboBoxFont, CB_GETLBTEXT, i,
(LPARAM) (LPCWSTR) szFontName);
        }
    }
}
```

```

        if (CompareStringEx(LOCALE_NAME_SYSTEM_DEFAULT,
NULL, fontName, -1, szFontName, -1, NULL, NULL, NULL) ==
CSTR_EQUAL) {
            doesHaveIT = TRUE;
            SendMessage(hComboBoxFont, CB_SETCURSEL, i,
NULL);
        }
    }
    if (!doesHaveIT) {
        SendMessage(hComboBoxFont, CB_GETLBTEXT, 0,
(LPPARAM) (LPCWSTR) fontName);
        if (createHkey(szKeyFont, fontName,
sizeof(fontName)) != 0)
            MessageBox(NULL, ERRCREATEHKEY, L"HKEY Error",
NULL);
    }
    doesHaveIT = FALSE;
}
}

//-----
// Считывает значения настроек из регистра цвета темы
//-----

void hCheckColor( TCHAR szKeyTheme[]) {
    BOOL doesHaveIT = FALSE;
    _TCHAR szWhiteTheme[10] = L"WHITE", szBlackTheme[10] =
L"BLACK";
    if (readHkey(szKeyTheme, themeName) == 65535) { //
функция возвращает -1
        if (createHkey(szKeyTheme, themeName, sizeof(themeName))
!= 0) {
            MessageBox(NULL, ERRCREATEHKEY, L"HKEY Error",
NULL);
        }
    }
    if (CompareStringEx(LOCALE_NAME_SYSTEM_DEFAULT, NULL,
themeName, -1, szWhiteTheme, -1, NULL, NULL, NULL) ==
CSTR_EQUAL) {
        THEME = gThemeColors[0];
        gFileColors[1] = gThemeColors[3];
        UPPERTHEME = gThemeColors[0];
    }
    else if (CompareStringEx(LOCALE_NAME_SYSTEM_DEFAULT, NULL,
themeName, -1, szBlackTheme, -1, NULL, NULL, NULL) ==
CSTR_EQUAL) {
        THEME = gThemeColors[1];
        gFileColors[1] = gThemeColors[0];
        UPPERTHEME = gThemeColors[2];
    }
    else {

```

```

        if (createHkey(szKeyTheme, szBlackTheme,
sizeof(szBlackTheme)) != 0) {
            MessageBox(NULL, ERRCREATEHKEY, L"HKEY Error",
NULL);
        }
    }
}

//-----
// Считывает значения настроек из регистра Размера шрифта
//-----

void hCheckSize(_TCHAR szKeyFontSize[]) {
    BOOL doesHaveIT = FALSE;
    if (readIntHkey(szKeyFontSize, &sFontSize) == -1) {    //
функция возвращает -1
        if (createIntHkey(szKeyFontSize, &sFontSize,
sizeof(sFontSize)) != 0) {
            MessageBox(NULL, ERRCREATEHKEY, L"HKEY Error",
NULL);
        }
    }
    else {
        int szFontSize = 0;
        char size[20] = { "13" };
        char str[3] = "13";
        for (int i = 0; i < 7; i++) {
            SendMessage(hComboBoxSize, CB_GETLBTEXT, i,
(LPARAM)size);
            str[0] = size[0];
            str[1] = size[2];
            szFontSize = atoi(str);
            if (sFontSize == szFontSize) {
                doesHaveIT = TRUE;
                SendMessage(hComboBoxSize, CB_SETCURSEL, i,
NULL);
            }
        }
        if (!doesHaveIT) {
            int index = SendMessage(hComboBoxSize, CB_GETCURSEL,
0, 0);
            SendMessage(hComboBoxSize, CB_GETLBTEXT, index,
(LPARAM)size);
            str[0] = size[0];
            str[1] = size[2];
            sFontSize = atoi(str);
            if (createIntHkey(szKeyFontSize, &sFontSize,
sizeof(sFontSize)) != 0) {
                MessageBox(NULL, ERRCREATEHKEY, L"HKEY Error",
NULL);
            }
        }
    }
}

```

```

    }
    doesHaveIT = FALSE;
}

}

//-----
// Считывает значения настроек из регистра количества файлов
//-----

void hCheckNumOfFiles(_TCHAR szKeyNumOfFiles[]) {
    BOOL doesHaveIT = FALSE;
    if (readIntHkey(szKeyNumOfFiles, &numOfFiles) == -1) {    //
        функция возвращает -1
        if (createIntHkey(szKeyNumOfFiles, &numOfFiles,
            sizeof(numOfFiles)) != 0) {
            MessageBox(NULL, ERRCREATEHKEY, L"HKEY Error",
            NULL);
        }
    }
    else {
        char strNumOfFiles[10] = "2";
        int szNumOfFiles = 0;
        for (int i = 0; i < 8; i++) {
            SendMessage(hNumOfFilesComboBox, CB_GETLBTEXT, i,
            (LPARAM) strNumOfFiles);
            szNumOfFiles = strNumOfFiles[0] - '0';
            szNumOfFiles = atoi(strNumOfFiles);
            if (numOfFiles == szNumOfFiles) {
                doesHaveIT = TRUE;
                SendMessage(hNumOfFilesComboBox, CB_SETCURSEL,
            i, NULL);
            }
        }
        if (!doesHaveIT) {
            numOfFiles = 2;
            if (createIntHkey(szKeyNumOfFiles, &numOfFiles,
            sizeof(numOfFiles)) != 0) {
                MessageBox(NULL, ERRCREATEHKEY, L"HKEY Error",
            NULL);
            }
        }
    }
}

//-----
// Считывает значения настроек из регистра, если их не находит
// создает нужный ключ и ставит значение по умолчанию
//-----

```

```

void hkeyCheck() {
    _TCHAR szKeyFont[] = _T("fontName"), szKeyTheme[] =
    _T("Theme"), szKeyFontSize[] = _T("fontSize"), szKeyNumOfFiles[]
    = _T("NumOfFiles");

    //-----
    ----- Шрифт ->
    hCheckFont(szKeyFont);

    //-----
    ----- Цвет темы ->
    hCheckColor(szKeyTheme);

    //-----
    ----- Размер шрифта ->
    hCheckSize(szKeyFontSize);

    //-----
    ----- Количество файлов ->
    hCheckNumOfFiles(szKeyNumOfFiles);
}

//-----
-----
// Функция обработки сообщений для главного окна
//-----
-----
LRESULT CALLBACK WindowProc(HWND hWnd, UINT uMsg, WPARAM wParam,
LPARAM lParam) {
    static size_t sCurrStr = 0; // Текущая строка для
отображения
    PAINTSTRUCT ps{};
    HDC hdc{};
    int mem = 0, again = 0, strLentgh = LINELENGTH;

    _tsetlocale(LC_ALL, _T(""));
    Buttons btnn{};
    btnn.Btnn1 = hButtonNext;
    btnn.Btnn2 = hButtonPrev;
    btnn.Btnn3 = hButtonOpenSettings;
    btnn.hProgressBar = hProgressBarForCheck;
    btnn.BtnnSearch = hText;
    btnn.hCompare = hCompare;

    switch (uMsg) {
        case WM_COMMAND: { // Обработка команд от элементов
управления
            switch (LOWORD(wParam)) {

                case BUTTON_CHANGETHHEME: {
                    changeTheme(hWnd, themeName, &THEME, &UPPERTHEME);

```

```

        break;
    }

    case BUTTON_COMPARE: {
        if (numOfFiles == numOfFilesOpened) { // Если все
        файлы открыты, начать сравнение
            StartDiffThread(hWnd, true, mapping, 0, btnn,
numOfFilesOpened, TRUE); // Запустить сравнение файлов
        }
        break;
    }

    case BUTTON_INFO: {
        MessageBox(hWnd, BUTTONINFOTEXT, L"О программе",
NULL);
        break;
    }

    case BUTTON_OPENSETTINGS: {
        ShowWindow(hSettingsWnd, TRUE); // Показать окно
настроек
        break;
    }

    case BUTTON_PREV: {
        if (numOfFilesOpened > 1) {
            StartDiffThread(hWnd, false, mapping, sCurrStr,
btnn, numOfFilesOpened); // Запустить сравнение назад
        }
        break;
    }

    case BUTTON_NEXT: {
        if (numOfFilesOpened > 1) {
            StartDiffThread(hWnd, true, mapping, sCurrStr,
btnn, numOfFilesOpened); // Запустить сравнение вперед
        }
        break;
    }

    case COMBOBOX_SIZE: {
        sFontSize = changeSize(hComboBoxSize, sFontSize,
wParam);
        break;
    }

    case COMBOBOX_FONT: {
        changeFont(hComboBoxFont, fontName, wParam,
sizeof(fontName));
        break;
    }
}
}

```

```

        refresh(numOfFiles, hChildDiff); // Обновить отображение
после команды
        SetFocus(hWnd);
        break;
    }

    case WM_CREATE: { // ----- Обработка
создания окна
        again = (int)wParam;
        if (again == 0) {
            createMainWindows(hWnd); //----- Создание
основных элементов управления
            hkeyCheck(); //----- Считывание
настроек из реестра, если они есть
        }
        createChildWindows(hWnd, numOfFiles); //--- Создание
дочерних окон для каждого файла
        break;
    }

    case WM_CTLCOLORSTATIC: {
        HDC hdcColor = (HDC)wParam;
        SetTextColor(hdcColor, gFileColors[1]);
        SetBkMode(hdcColor, TRANSPARENT);
        return (LONG) (HBRUSH)GetStockObject(NULL_BRUSH);
    }

    case WM_PAINT: { // Обработка перерисовки окна
        hdc = paintCase(hWnd, UPPERTHEME, ps);
        break;
    }

    case WM_SIZE: { // Обработка изменения размера окна
        size_t bigNum = ((bigger(mapping, numOfFiles)) /
LINELENGTH) + 1;
        sizeChangedCase(hWnd, numOfFiles, bigNum, sCurrStr,
sFontSize);
        break;
    }

    case WM_KEYDOWN: { // Обработка нажатия клавиш
        if (mapping) {
            size_t bigNum = (bigger(mapping, numOfFiles)) /
LINELENGTH;
            if (((bigger(mapping, numOfFiles)) - (LINELENGTH *
bigNum)) > 0) bigNum++;
            // Обработка нажатой клавиши
            sCurrStr = keydownCase(wParam, ScrollIsON, hWnd,
numOfFiles, bigNum, sCurrStr, sFontSize);
        }
        break;
    }
}

```

```

        case WM_MOUSEWHEEL: { // Обработка прокрутки колесиком мыши
            if (ScrollIsON) {
                short zDelta      = (short)(0xffff & (wParam >> 16));
                int amount        = 4;
                size_t bigNum     = (bigger(mapping, numOfFile)) /
LINELENGTH;
                if (((bigger(mapping, numOfFile)) - (LINELENGTH *
bigNum)) > 0) bigNum++;
                if (zDelta > 0) { // Прокрутка вверх
                    if (sCurrStr > amount) sCurrStr -= amount;
                    else if (sCurrStr > 0 && sCurrStr < amount + 1)
sCurrStr = 0;
                }
                else { // Прокрутка вниз
                    if (sCurrStr < (bigNum -
(whatSize(hChildDiff[0], sFontSize))) - (amount - 1))
                    {
                        sCurrStr += amount;
                    }
                    else sCurrStr = (bigNum -
(whatSize(hChildDiff[0], sFontSize)));
                }

                // Отправка сообщения дочерним окнам о текущей
строке
                SendMessage(hChildDiff[0], WM_USER, sCurrStr, 0);
                for (int y = 0; y < numOfFile; y++) {
                    setScrlInformation(hChildDiff[y], sCurrStr,
(bigNum - 1) - (whatSize(hChildDiff[0], sFontSize))); //
Установка информации для полос прокрутки
                }
                refresh(numOfFile, hChildDiff); // Обновить
отображение
            }
            break;
        }

        case WM_USER: { // Пользовательское сообщение для обновления
текущей строки
            sCurrStr = (size_t)wParam;
            size_t bigNum = (bigger(mapping, numOfFile)) /
LINELENGTH;
            for (int y = 0; y < numOfFile; y++) {
                setScrlInformation(hChildDiff[y], sCurrStr, bigNum -
whatSize(hChildDiff[0], sFontSize)); // Установка информации для
полос прокрутки
            }
            refresh(numOfFile, hChildDiff); // Обновить отображение
            break;
        }
    }
}

```

```

        case WM_USER + 1: { // Пользовательское сообщение для
отображения сообщения о различиях
            size_t bigNum = (bigger(mapping, numOfFile)) /
LINELENGTH;
            if (((bigger(mapping, numOfFile)) - (LINELENGTH *
bigNum)) > 0) bigNum++;
            size_t bigMem = 0;
            mem = (int)wParam;
            if (mem > 0) bigMem = (size_t)wParam;
            else if (mem < -2) { bigMem = (size_t)wParam; }

            if (mem == -1) { MessageBox(hWnd, NODIFFERENCE,
L"Сообщение", NULL); }
            else if (mem == -2) { break; }
            else {
                if (bigNum > whatSize(hChildDiff[0], sFontSize)) {
                    sCurrStr = bigMem;
                    if (sCurrStr > bigNum - (whatSize(hChildDiff[0],
sFontSize)))
                        sCurrStr = bigNum - (whatSize(hChildDiff[0],
sFontSize));
                    for (int y = 0; y < numOfFile; y++) {
                        setScrlInformation(hChildDiff[y], sCurrStr,
bigNum - (whatSize(hChildDiff[0], sFontSize))); // Установка
информации для полос прокрутки
                    }
                    updateStr(numOfFile, hChildDiff, sCurrStr); //
Обновить строки для отображения
                }
            }
            break;
        }

        case WM_USER + 2: { // Пользовательское сообщение для
установки позиции прокрутки HORZ
            int pos = (int)wParam;
            for (int y = 0; y < numOfFile; y++) {
                SetScrollPos(hChildDiff[y], SB_HORZ, pos, TRUE);
            }
            refresh(numOfFile, hChildDiff); // Обновить отображение
            break;
        }

        case WM_USER + 3: { // Пользовательское сообщение для
установки текущей строки
            sCurrStr = (size_t)wParam;
            break;
        }

        case WM_USER + 4: { // Пользовательское сообщение для
отображения сообщения о состоянии различий
            size_t bigMem = 0;

```

```

        mem = (int)wParam;
        if (mem > 0) bigMem = (size_t)wParam;

        if (mem == -1) { MessageBox(hWnd, NODIFFERENCEINFILES,
L"Похожи", NULL); }
        else if (mem == -2) { break; }
        else { MessageBox(hWnd, HAVEDIFFERENCEINFILES,
L"Разные", NULL); }
        break;
    }

    case WM_USER + 5: { // Пользовательское сообщение для
установки позиции прокрутки VERT
        int pos = (int)wParam;
        for (int y = 0; y < numOfFile; y++) {
            SetScrollPos(hChildDiff[y], SB_VERT, pos, TRUE);
        }
        refresh(numOfFile, hChildDiff); // Обновить отображение
        break;
    }

    case WM_CLOSE: {
        if (isSearching()) {
            if (MessageBox(GetParent(hWnd), STOPSEARCHWINDOW,
L"Сообщение",
                MB_OKCANCEL | MB_ICONWARNING | MB_SYSTEMMODAL)
== IDCANCEL) {
                break;
            }
        }
        DestroyWindow(hWnd);
    }

    case WM_DESTROY: // Обработка закрытия окна
        PostQuitMessage(0);
        break;

    default:
        return DefWindowProc(hWnd, uMsg, wParam, lParam);
    }
    return 0;
}

//-----
// Функция обработки сообщений для дочерних окон
//-----

LRESULT CALLBACK ChildWindowProc(HWND hWnd, UINT uMsg, WPARAM
wParam, LPARAM lParam) {
    OPENFILENAME fileFirst{}, fileSecond{};
    char szFileFirst[300] {}, szFileSecond[300] {};

```

```

    PAINTSTRUCT ps{};
    HDC hdc{};
    static size_t sCurrStr = 0;
    int windowID          = (int)GetMenu(hWnd); windowID -=
100; // Получение ID окна
    static int sX          = 15; // Позиция по горизонтали
    BOOL bottom           = FALSE;

    bottom = doNeedHorz(hWnd, sFontSize); // Проверка, нужно ли
показывать горизонтальную прокрутку
    #pragma region HORZSCROLL

    if (bottom) {
        ShowScrollBar(hWnd, SB_HORZ, TRUE); // Показать
прокрутку
    }
    else {
        sX = 15;
        //ShowWindow(hHorzScrollBar[windowID], SW_HIDE);
        ShowScrollBar(hWnd, SB_HORZ, FALSE); // скрыть прокрутку
    }
    #pragma endregion

    switch (uMsg) {
    case WM_COMMAND: {
    case BUTTON_FILEOPEN: // Обработка нажатия кнопки "Открыть
файл"
        EnableWindow(hButtonFileCreate[windowID], FALSE);
        if (isSearching()) {
            if (MessageBox(GetParent(hWnd), STOPSEARCHOPENFILE,
L"Сообщение",
                MB_OKCANCEL | MB_ICONWARNING | MB_SYSTEMMODAL)
== IDCANCEL) {
                EnableWindow(hButtonFileCreate[windowID], TRUE);
                break;
            }
        }

        if (mapping[windowID]) { // Если файл уже открыт,
закрыть его
            fileMappingClose(mapping[windowID]);
            mapping[windowID] = {};
            numOfFilesOpened--;
        }

        if (!mapping[windowID]) { // Если файл не открыт,
открыть его
            ZeroMemory(&fileFirst, sizeof(fileFirst));
            fileFirst.lStructSize = sizeof(fileFirst);
            fileFirst.lpstrFile = (LPWSTR)szFileFirst;
            fileFirst.lpstrFile[0] = '\\0';
            fileFirst.nMaxFile = sizeof(szFileFirst);

```

```

        fileFirst.lpstrFilter = L"All\0*.*\0Text\0*.TXT\0";
        fileFirst.nFilterIndex = 1;
        fileFirst.lpstrFileTitle = NULL;
        fileFirst.nMaxFileTitle = 0;
        fileFirst.lpstrInitialDir = NULL;
        fileFirst.Flags = OFN_PATHMUSTEXIST |
OFN_FILEMUSTEXIST;
        GetOpenFileName(&fileFirst); // Открыть диалог
        выбора файла
        SendMessage(hBoxNameOfFile[windowID],
CB_DELETESTRING, 0, 0);
        SendMessage(hBoxNameOfFile[windowID], CB_ADDSTRING,
0, (LPARAM)fileFirst.lpstrFile);
        SendMessage(hBoxNameOfFile[windowID], CB_SETCURSEL,
0, 0);

        mapping[windowID] =
fileMappingCreate(fileFirst.lpstrFile); // Создать отображение
файла в памяти
        if (mapping[windowID]) {
            numFilesOpened++;
            size_t bigNum = bigger(mapping, numFiles);
            if (doNeedVert(bigNum, whatSize(hWnd,
sFontSize))) {
                ShowScrollBar(hWnd, SB_VERT, TRUE);
                ScrollIsON = TRUE;
            }
            else {
                ShowScrollBar(hWnd, SB_VERT, FALSE);
                ScrollIsON = FALSE;
            }
        }
        sCurrStr = 0;
        SendMessage(GetParent(hWnd), WM_USER + 3, sCurrStr, 0);
// Установить текущую строку в главном окне
        SendMessage(GetParent(hWnd), WM_USER + 5, 0, 0);

        refresh(numFiles, hChildDiff); // Обновить отображение
        EnableWindow(hButtonFileCreate[windowID], TRUE);
        SetFocus(GetParent(hWnd));
        break;
    }

    case WM_PAINT: { // Обработка перерисовки дочернего окна
        ForTextOut tOut{};
        hdc = BeginPaint(hWnd, &ps);
        RECT rect{};
        HBRUSH brush = CreateSolidBrush(THEME); // Создание
кисти для закраски
        HFONT hFontOriginal{}, hFont = CreateFont(sFontSize,
sFontSize / 2, 0, 0, FW_DONTCARE, FALSE, FALSE, FALSE,

```

```

ANSI_CHARSET, OUT_TT_PRECIS, CLIP_DEFAULT_PRECIS,
DEFAULT_QUALITY, DEFAULT_PITCH | FF_DONTCARE,
(LPCWSTR)fontName);
    GetClientRect(hWnd, &rect);
    if (bottom) rect.bottom - 18;
    HDC memDC = CreateCompatibleDC(hdc);
    HBITMAP bmp = CreateCompatibleBitmap(hdc, rect.right -
rect.left, rect.bottom - rect.top);
    HBITMAP oldBmp = (HBITMAP)SelectObject(memDC, bmp);
    FillRect(memDC, &rect, brush); // Закраска фона
    hFontOriginal = (HFONT)SelectObject(memDC, hFont); //
Установка шрифта

    if (mapping[windowID]) { // Если файл открыт, вывести
его содержимое
        tOut.currFile = windowID;
        tOut.currStr = sCurrStr;
        tOut.fontSize = sFontSize;
        tOut.numOfFiles = numOfFiles;
        tOut.numOfFilesOpened = numOfFilesOpened;
        tOut.hdc = memDC;
        tOut.x = sX;

        txtOut(mapping, hWnd, tOut); // Вывод текста файла
    }
    // Перенос изображения из памяти в окно
    BitBlt(hdc, 0, 0, rect.right - rect.left, rect.bottom -
rect.top, memDC, 0, 0, SRCCOPY);
    SelectObject(memDC, oldBmp);
    DeleteObject(bmp);
    DeleteDC(memDC);
    EndPaint(hWnd, &ps);
    DeleteObject(hFont);
    DeleteObject(brush);
    return 0;
}

case WM_DESTROY: { // Обработка закрытия дочернего окна
    if (mapping[windowID]) {
        fileMappingClose(mapping[windowID]); // Закрыть
отображение файла
        mapping[windowID] = {};
    }
    break;
}

case WM_MOUSEMOVE: {
    POINT pt{};
    GetCursorPos(&pt);
    ScreenToClient(hBoxNameOfFile[windowID], &pt);
    RECT rc{};
    GetClientRect(hBoxNameOfFile[windowID], &rc);

```

```

        if (PtInRect(&rc, pt)) {
            int length = SendMessage(hBoxNameOfFile[windowID],
WM_GETTEXTLENGTH, 0, 0);
            TCHAR* buffer = new TCHAR[length + 1]{};
            SendMessage(hBoxNameOfFile[windowID], WM_GETTEXT,
length + 1, (LPARAM)buffer);
            ti[windowID].lpszText = buffer;
            SendMessage(hTooltip[windowID], TTM_UPDATETIPTTEXT,
0, (LPARAM)&ti[windowID]);
            SendMessage(hTooltip[windowID], TTM_TRACKPOSITION,
0, (LPARAM)MAKELONG(pt.x, pt.y));
            SendMessage(hTooltip[windowID], TTM_TRACKACTIVATE,
TRUE, (LPARAM)&ti[windowID]);
            if (buffer != NULL)
                delete[] buffer;
        }
        else {
            SendMessage(hTooltip[windowID], TTM_TRACKACTIVATE,
FALSE, (LPARAM)&ti[windowID]);
        }
        break;
    }

    case WM_VSCROLL: { // Обработка вертикальной прокрутки
        if (ScrollIsON) {
            RECT rect{};
            GetClientRect(hWnd, &rect);
            SCROLLINFO scrInfo{};
            BOOL notStr = TRUE;
            size_t bigNum = (bigger(mapping, numOfFile)) /
LINELENGTH;
            if (((bigger(mapping, numOfFile)) - (LINELENGTH *
bigNum)) > 0) bigNum++;

            scrInfo.cbSize = sizeof(SCROLLINFO);
            scrInfo.fMask = SIF_ALL;
            GetScrollInfo(hWnd, SB_VERT, &scrInfo);
            int currentPos = scrInfo.nPos;

            switch (LOWORD(wParam)) {
                case SB_LINEUP:
                    if (sCurrStr > 0) sCurrStr--;
                    notStr = FALSE;
                    break;
                case SB_LINEDOWN:
                    sCurrStr++;
                    if (sCurrStr > bigNum -
(whatSize(hChildDiff[0], sFontSize)))
                        sCurrStr = (bigNum -
(whatSize(hChildDiff[0], sFontSize)));
                    notStr = FALSE;
            }
        }
    }
}

```

```

        break;
    case SB_PAGEUP:
        sCurrStr -= whatSize(hWnd, sFontSize);
        notStr = FALSE;
        break;
    case SB_PAGEDOWN:
        sCurrStr += whatSize(hWnd, sFontSize);
        notStr = FALSE;
        break;
    case SB_THUMBTRACK:
        scrInfo.nPos = scrInfo.nTrackPos;
        break;
    default:
        return 0;
        break;
}

scrInfo.fMask = SIF_POS;
SetScrollInfo(hWnd, SB_VERT, &scrInfo, TRUE);
GetScrollInfo(hWnd, SB_VERT, &scrInfo);
if (notStr) {
    float cur = scrInfo.nPos;
    float max = (scrInfo.nMax);
    sCurrStr = vertScroll(cur, max, mapping,
numOfFiles, (whatSize(hWnd, sFontSize))); // Вычислить текущую
строку
        SendMessage(GetParent(hWnd), WM_USER + 5, cur,
0);
        SendMessage(GetParent(hWnd), WM_USER + 3,
sCurrStr, 0);
    }
    else {

        if (sCurrStr > bigNum - (whatSize(hWnd,
sFontSize)))
            sCurrStr = bigNum - whatSize(hWnd,
sFontSize);
        SendMessage(GetParent(hWnd), WM_USER, sCurrStr,
0); // Обновить главное окно
    }
}
return 0;
}

case WM_HSCROLL: { // Обработка горизонтальной прокрутки
    RECT rect{};
    GetClientRect(hWnd, &rect);
    SCROLLINFO scrInfo{};
    float yLength = rect.right - rect.left;
    scrInfo.cbSize = sizeof(SCROLLINFO);
    scrInfo.fMask = SIF_ALL;
    GetScrollInfo(hWnd, SB_HORZ, &scrInfo);

```

```

switch (LOWORD(wParam)) {
case SB_LINERIGHT:
    scrInfo.nPos += 1;
    break;
case SB_LINELEFT:
    scrInfo.nPos -= 1;
    break;
case SB_THUMBTRACK:
    scrInfo.nPos = scrInfo.nTrackPos;
    break;
default:
    return 0;
    break;
}
scrInfo.fMask = SIF_POS;
SetScrollInfo(hWnd, SB_HORZ, &scrInfo, TRUE);
GetScrollInfo(hWnd, SB_HORZ, &scrInfo);
sX = 15;
float width = sFontSize * (LINELENGTH * 2 + 12);
sX = sX - ((double)((width - yLength) / 100) *
(scrInfo.nPos));
SendMessage(GetParent(hWnd), WM_USER + 2, scrInfo.nPos,
0); // Обновить позицию прокрутки в главном окне
return 0;
}

case WM_SIZE: { // Обработка изменения размера дочернего
окна
    sX = 15;
    RECT rect{};
    size_t bigNum = bigger(mapping, numOfFile);
    GetClientRect(hWnd, &rect);
    SCROLLINFO scrInfo{};
    scrInfo.nTrackPos = 0;
    scrInfo.fMask = SIF_POS;
    SetScrollInfo(hWnd, SB_HORZ, &scrInfo, TRUE); // Сброс
позиции прокрутки
    if (mapping[windowID]) {
        if (doNeedVert(bigNum, whatSize(hWnd, sFontSize))) {
            ShowScrollBar(hWnd, SB_VERT, TRUE);
            ScrollIsON = TRUE;
        }
        else {
            ShowScrollBar(hWnd, SB_VERT, FALSE);
            ScrollIsON = FALSE;
        }
    }
    break;
}

case WM_USER: // Пользовательское сообщение для установки
текущей строки

```

```

        sCurrStr = (size_t)wParam;
        break;

    case WM_ERASEBKGND: // Предотвращение мерцания при
перерисовке
        return (LRESULT)1;

    default:
        return DefWindowProc(hWnd, uMsg, wParam, lParam);
    }
    return 0;
}

//-----
// Функция обработки сообщений для окна настроек
//-----

LRESULT CALLBACK SettingsWindowProc(HWND hWnd, UINT uMsg, WPARAM
wParam, LPARAM lParam) {
    PAINTSTRUCT ps{};
    HDC hdc{};
    _TCHAR szKeyNumOfFiles[] = _T("NumOfFiles");
    HWND hButtonClose, hButtonCancel;
    static int sNumOfFilesMem = numOfFiles;
    switch (uMsg) {

    case WM_CREATE: {
        hButtonClose = CreateWindowEx(0, L"BUTTON", L"X",
            WS_TABSTOP | WS_VISIBLE | WS_CHILD |
BS_DEFPUSHBUTTON,
            300, 10, 30, 25, hWnd, (HMENU)BUTTON_CLOSESETTINGS,
            (HINSTANCE)GetWindowLong(hWnd, GWLP_HINSTANCE),
            NULL);
        hButtonCancel = CreateWindowEx(0, L"BUTTON",
L"Отменить",
            WS_TABSTOP | WS_VISIBLE | WS_CHILD |
BS_DEFPUSHBUTTON,
            30, 80, 130, 25, hWnd, (HMENU)BUTTON_CANCELSETTINGS,
            (HINSTANCE)GetWindowLong(hWnd, GWLP_HINSTANCE),
            NULL);
    }

    case WM_COMMAND: { // Обработка команд от элементов
управления
        switch (LOWORD(wParam)) {
            case SETTINGS_BUTTON: {
                changeNumOfFiles(numOfFiles, hChildDiff,
hButtonFileCreate, GetParent(hWnd)); // Изменить количество
файлов
                numOfFiles = sNumOfFilesMem;
            }
        }
    }
    }
}

```

```

        SendMessage(GetParent(hWnd), WM_CREATE, 1, NULL); //
Пересоздать главное окно с новым количеством файлов
        SendMessage(GetParent(hWnd), WM_SIZE, 0, 0);
        SendMessage(GetParent(hWnd), WM_USER + 3, 0, 0); //
Установить текущую строку в 0
        numOfFilesOpened = 0; // Сброс количества открытых
файлов
        if (createIntHkey(szKeyNumOfFiles, &numOfFiles,
sizeof(numOfFiles)) != 0) {
            MessageBox(NULL, ERRRCREATEHKEY, L"HKEY Error",
NULL);
        }
        refresh(numOfFiles, hChildDiff); // Обновить
отображение
        ShowWindow(hWnd, FALSE); // Скрыть окно настроек
        SetFocus(GetParent(hWnd)); // Установить фокус на
главное окно
        break;
    }

    case BUTTON_CANCELSETTINGS: {
        ShowWindow(hWnd, FALSE); // Скрыть окно настроек
        SetFocus(GetParent(hWnd)); // Установить фокус на
главное окно
        break;
    }

    case BUTTON_CLOSESETTINGS: {
        ShowWindow(hWnd, FALSE); // Скрыть окно настроек
        SetFocus(GetParent(hWnd)); // Установить фокус на
главное окно
        break;
    }

    case SETTINGS_COMBOBOX: {
        if (HIWORD(wParam) == CBN_SELCHANGE) { // Изменение
выбора в комбобоксе
            if (LOWORD(wParam) == SETTINGS_COMBOBOX) {
                char size[10] = { "2" };
                int index = SendMessage(hNumOfFilesComboBox,
CB_GETCURSEL, 0, 0);
                SendMessage(hNumOfFilesComboBox,
CB_GETLBTEXT, index, (LPARAM)size);
                sNumOfFilesMem = size[0] - '0'; //
Установить новое количество файлов
                break;
            }
        }
    }
}
}
}
}

```

```

        case WM_PAINT: { // Обработка перерисовки окна
            hdc = BeginPaint(hWnd, &ps);
            HBRUSH brush = CreateSolidBrush(UPPERTHEME); // Создание
кисти для закраски
            RECT rect{};
            GetClientRect(hWnd, &rect);
            FillRect(hdc, &rect, brush); // Закраска верхней части
            EndPaint(hWnd, &ps);
            DeleteObject(brush);
            return 0;
        }

        case WM_DESTROY: { // Обработка закрытия окна настроек
            break;
        }
        case WM_SIZE: { // Обработка изменения размера окна настроек
            break;
        }
        default:
            return DefWindowProc(hWnd, uMsg, wParam, lParam);
        }
        return 0;
}

```

Файл «FilePrinting.cpp»

```

#define _CRT_SECURE_NO_WARNINGS
#include "Header.h"

/// <summary>
/// Печатает адрес выводимых символов
/// </summary>
/// <param name="hdc">hdc экрана</param>
/// <param name="currStr">Номер текущей строки</param>
/// <param name="strLength">Количество символов из файла на одну
строку</param>
/// <param name="x">Координата x</param>
/// <param name="y">Координата y</param>
/// <returns></returns>
size_t adrPrinting(HDC hdc, size_t currStr, int strLength, int
x, int y) {
    wchar_t lps[20] = { 0 };
    SetTextColor(hdc, gFileColors[1]);
    swprintf(lps, (sizeof(lps) / sizeof(lps[0])), L"%zX",
(currStr * strLength));
    currStr++;
    TextOut(hdc, x, y, lps, 20);
    return currStr;
}

```

```

/// <summary>
/// Формирует строку нужного цвета
/// </summary>
/// <param name="strForHex">Строка с 16-ым отображением</param>
/// <param name="strForData">Строка с обычным
отображением</param>
/// <param name="data">Структура с данными файлов</param>
/// <param name="fileNum">Номер файла</param>
/// <param name="currFile">Номер текущего файла</param>
/// <param name="u">Номер символа в строке</param>
/// <param name="k">Номер символа в прочитанном буфере</param>
void stringMaker(char strForHex[9][50], char strForData[9][18],
DataPoint* pData, int fileNum, int currFile, int u, int k) {
    char strJohn[2]{};
    symbCheckSec(strForHex[fileNum], pData[currFile], u, k,
strJohn);
    symbCheck((unsigned char*)strForData[fileNum],
pData[currFile], u, k);
}

```

```

/// <summary>
/// Формирует пустую строку
/// </summary>
/// <param name="strForHex">Строка с 16-ым отображением</param>
/// <param name="strForData">Строка с обычным
отображением</param>
/// <param name="u">Номер символа в строке</param>
/// <param name="numOfFiles">Количество файлов</param>
void stringBlanker(char strForHex[9][50], char
strForData[9][18], int u, int numOfFiles) {
    for (int j = 0; j <= numOfFiles; j++) {
        symbCheckThird(strForHex[j], u);
        strForData[j][u] = ' ';
    }
}

```

```

/// <summary>
/// Отрисовывает строки на экран в соответствии с их цветами
/// </summary>
/// <param name="strForHex">Строка с 16-ым отображением</param>
/// <param name="strForData">Строка с обычным
отображением</param>
/// <param name="y">Координата y</param>
/// <param name="txtData">Структура с основными данными для
отрисовки строчек</param>
void stringPrint(char strForHex[9][50], char strForData[9][18],
int y, ForTextOut txtData) {
    int numOfFiles          = txtData.numOfFiles;
    int  numOfFilesOpened   = txtData.numOfFilesOpened;
    HDC  hdc                = txtData.hdc;

```

```

    int  fontSize          = txtData.fontSize;
    int  x                  = txtData.x;
    size_t size1           = LINELENGTH + 3;
    wchar_t* dst1          = new wchar_t[size1];
    int  wideX             = LINELENGTH / 8;
    size_t outSize;

    if (numOfFilesOpened == 1) {
        SetTextColor(hdc, gFileColors[1]);
        if (strForHex[0])
            TextOutA(hdc, x, y, strForHex[0], 50);
        x = x + fontSize * (13 * wideX + 1);

        mbstowcs_s(&outSize, dst1, size1, strForData[0], size1
- 1);

        SetTextColor(hdc, gFileColors[1]);
        if (strForData[0])
            TextOutW(hdc, x, y, dst1, outSize);
    }
    else {
        for (int h = 0; h <= numOfFiles; h++) {
            SetTextColor(hdc, gFileColors[h]);
            if (strForHex[h])
                TextOutA(hdc, x, y, strForHex[h], 50);
        }
        x = x + fontSize * (13 * wideX + 1);

        for (int h = 0; h <= numOfFiles; h++) {
            mbstowcs_s(&outSize, dst1, size1, strForData[h],
size1 - 1);

            SetTextColor(hdc, gFileColors[h]);
            if (strForData[h])
                TextOutW(hdc, x, y, dst1, outSize);
        }
    }
    if(dst1!=NULL)
        delete[] dst1;
}

/// <summary>
/// Сравнивает символы всех открытых файлов
/// </summary>
/// <param name="strForHex">Строка с 16-ым отображением</param>
/// <param name="strForData">Строка с обычным
отображением</param>
/// <param name="cycleData">Структура с данными основного цикла
сравнения файлов</param>
/// <param name="data">Структура с данными файлов</param>
void mainCycle(char strForHex[9][50], char strForData[9][18],
ForTextFormingCycle cycleData, DataPoint* pData) {
    BOOL haveBeenBefore = FALSE;
    int currFile        = cycleData.currFile;

```

```

int u                = cycleData.u;
int k                = cycleData.k;
int numOfFiles       = cycleData.numOfFiles;
size_t endFileNum    = cycleData.endFileNum;

if (pData[currFile].fsize <= endFileNum) {
    stringBlanker(strForHex, strForData, u, numOfFiles);
}
else {
    stringBlanker(strForHex, strForData, u, numOfFiles);

    for (int j = 0; j < numOfFiles; j++) {
        if (j != currFile && !pData[j].fsize == 0)
        {
            if ((pData[currFile].data[k]
==pData[j].data[k]) && (!haveBeenBefore) && (pData[j].fsize >=
endFileNum)) {
                if (currFile < j) {
                    // currfile ... Если символы похожи
                    // и цвет надо брать с файла
                    stringMaker(strForHex,
strForData,pData, currFile + 1, currFile, u, k);
                    haveBeenBefore = TRUE;
                }
                else {
                    //j ... Если символы похожи
                    // и цвет надо брать с текущего
                    stringMaker(strForHex,
strForData,pData, j + 1, currFile, u, k);
                    haveBeenBefore = TRUE;
                }
            }
            else {
                //0 ... Если символы не похожи
                stringMaker(strForHex, strForData,pData,
0, currFile, u, k);
            }
        }
        else {
            //0 ... Если сравниваемый файл закончился
            stringMaker(strForHex, strForData,pData, 0,
currFile, u, k);
        }
    }
}

/// <summary>

```

```

/// Создание буфера 128 кбайт для просмотра данных
/// </summary>
/// <param name="pMapping">Указатель на структуру с маппингом
файлов</param>
/// <param name="currPosInFile">В каком по счету буфере мы
находимся</param>
/// <param name="member">Структура куда будет записываться
буфер</param>
void buffMaker(FileMapping* pMapping, int currPosInFile,
DataPoint& member) {
    size_t offset = 64 * 1024; // размер одного буфера 64 кбит
    unsigned char* dataPointer = nullptr;
    member.fsize = pMapping->fsize;
    member.lastPoint = deathPoint(offset * currPosInFile,
offset, pMapping->fsize); // DeathPoint - точка до которой мы
может "мапить" данные
    if ((member.lastPoint) <= (currPosInFile * offset)) or
currPosInFile == 0) {
        dataPointer = mapFilePart(pMapping->hMapping, offset *
currPosInFile, offset, pMapping->fsize);
        member.mappedData = dataPointer; // Сохраняем
указатель

        for (size_t i = 0; i < member.lastPoint; i++) {
            member.data[i] = dataPointer[i]; // Заполнение
буфера
        }
    }
    currPosInFile++;
    member.lastPoint = deathPoint(offset * (currPosInFile),
offset, pMapping->fsize); // DeathPoint - точка до которой мы
может "мапить" данные
    if ((member.lastPoint <= ((currPosInFile) * offset)) or
currPosInFile == 0) {
        dataPointer = mapFilePart(pMapping->hMapping, offset *
(currPosInFile), offset, pMapping->fsize);
        if (dataPointer) {
            for (size_t i = 0; i < member.lastPoint; i++) {
                member.data[i + offset] = dataPointer[i]; //
Заполнение буфера
            }
        }
    }
    if (dataPointer)
        dataPointer = nullptr;
}

/// <summary>
/// Основная функция отрисовки и сравнения файлов
/// </summary>

```

```

/// <param name="numOfShownedStr">Количество строк на
экране</param>
/// <param name="k">Номер символа в прочитанном буфере</param>
/// <param name="data">Структура с данными файлов</param>
/// <param name="Greater">Размер большего из всех файлов</param>
/// <param name="tOut">Структура с переменными для вывода текста
на экран</param>
void diffTxtOut(int numOfShownedStr, int k, DataPoint* data,
size_t Greater, ForTextOut tOut) {

    #pragma region variables
    std::locale::global(std::locale(""));
    char strJohn[2]{};
    char strForHex[MAXNUMOFFILES][50]{};
    char strForData[MAXNUMOFFILES][18]{};
    ForTextOut txtData{}; ForTextFormingCycle cycleData{};

    cycleData.numOfFiles          = tOut.numOfFiles;

    txtData.numOfFiles            = tOut.numOfFiles;
    txtData.numOfFilesOpened      = tOut.numOfFilesOpened;
    txtData.hdc                   = tOut.hdc;
    txtData.fontSize              = tOut.fontSize;
    size_t endFileNum             = 0;
    HDC hdc                       = tOut.hdc;
    size_t currStr                = tOut.currStr;
    int x                         = tOut.x;
    int fontSize                  = tOut.fontSize;
    int currFile                  = tOut.currFile;
    int numOfFiles               = tOut.numOfFiles;
    int numOfFilesOpened          = tOut.numOfFilesOpened;
    int strLength                 = LINELENGTH, y = 36, SS = 16,
tempX = x;
    BOOL haveBeenBefore          = FALSE;
    size_t maxStr                 = Greater / strLength;
    if ((Greater - (strLength * maxStr)) > 0) maxStr++;
    int counter                   = maxStr == 0 ? 1 :
log10(maxStr);
    SetBkMode(hdc, TRANSPARENT);
    #pragma endregion

    for (int i = 0; (i < numOfShownedStr) && (currStr < maxStr);
i++) {
//-----
---- адрес данных->
        currStr = adrPrinting(hdc, currStr, strLength, x, y);
        x = x + (fontSize * (counter - counter / 2 + 1));
//-----
---- занос строки во временный буффер->
        for (int u = 1; u <= strLength; u++) {
            haveBeenBefore = FALSE;

```

```

        endFileNum = (size_t)(currStr - 1) * strLength + u
- 1;

        for (int j = 0; j <= numFiles; j++) {
            strForHex[j][0] = ' ';
            strForData[j][0] = ' ';
        }
        strForHex[1][0] = '|';
        strForData[1][0] = '|';
//-----
---- Цикл сравнения символов между файлами
        cycleData.currFile = currFile;
        cycleData.u = u;
        cycleData.k = k;
        cycleData.endFileNum = endFileNum;
        mainCycle(strForHex, strForData, cycleData, data);
        k++;
    }
//-----
---- вывод строк разных цветов на экран->

        txtData.x = x;
        stringPrint(strForHex, strForData, y, txtData);
        x = tempX;
        y = y + fontSize + 3;
    }
}

/// <summary>
/// Функция заполнения буфера (при необходимости)
/// </summary>
/// <param name="pMapping">Указатель на структуру с маппингом
файлов</param>
/// <param name="hWnd">Указатель на окно отрисовки</param>
/// <param name="tOut">Структура с переменными для вывода текста
на экран</param>
/// <returns></returns>
int txtOut(FileMapping* pMapping[], HWND hWnd, ForTextOut tOut)
{
    size_t Greater = bigger(pMapping, tOut.numFilesOpened);
    int strLength = LINELENGTH;
    const size_t offset = 64 * 1024;
    int numShownStr = whatSize(hWnd, tOut.fontSize);

    size_t k = tOut.currStr * strLength;

    size_t currPosInFile = k / offset;
    size_t check = tOut.currStr * strLength;
    k = k - (currPosInFile * offset);

    static DataPoint sData[MAXNUMOFFILES]{};

```

```

static int sPrevPos = 0;
while (check > (offset * 2)) {
    check -= offset;
}

for (int h = 0; h < tOut.numOfFiles; h++) {
    if (!pMapping[h]) sData[h].fsize = 0;
}
if (check < (offset * 1.5) && check != 0 && (sPrevPos ==
currPosInFile)) {
    diffTxtOut(numOfShownedStr, k, sData, Greater, tOut);
}

else {
    for (int h = 0; h < tOut.numOfFilesOpened; h++) {
        if (pMapping[h]) buffMaker(pMapping[h],
currPosInFile, sData[h]);
    }
    diffTxtOut(numOfShownedStr, k, sData, Greater, tOut);
}
sPrevPos = currPosInFile;
return 0;
}

```

Приложение В

(Обязательное)

Бизнес-проект «программа для побитового сравнения множества файлов
большого объема»

Презентация

					ПГУ1.090301.10.001 ПЗ	Лист
						108
Изм.	Лист	№ докум.	Подпись	Дата		



Бизнес-проект: «Программа для побитового сравнения множества файлов большого объема»

Выполнил: Кожняков Д.М. 21ВВП1

Пензенский Государственный Университет, 2025 год.



Актуальность и цель проекта

Объемы цифровых данных растут, их целостность критически важна. Существующие решения не справляются с большими файлами и побитовым анализом.

-  Растущие данные
Необходимость верификации информации.
-  Ограничения аналогов
Проблемы с большими файлами и детальным анализом.
-  Цель проекта
Разработка ПО для точного побитового сравнения.

Обзор аналогов и выбор инструментов

Существующие решения, такие как Total Commander и WinHex, имеют ограничения по размеру файлов и количеству сравниваемых объектов.

Ограничения аналогов

- Не поддерживают терабайтные файлы.
- Ограничены 2 файлами для сравнения.
- Нет детальной визуализации.

Выбор инструментов

- **C++:** Максимальная производительность.
- **Windows API:** Высокая отзывчивость UI.
- **Visual Studio 2019:** Мощная среда разработки.

Архитектура и технологии

Программа использует модульную архитектуру для гибкости и масштабирования. Ключевые решения обеспечивают высокую производительность.

Модульная архитектура

Гибкость и масштабирование.

Многопоточность

Предотвращение зависаний UI.



Memory-mapped files

Быстрый доступ к данным.

"Скользящее окно"

Обработка больших файлов.

Пользовательский интерфейс и визуализация

Интерфейс обеспечивает наглядное отображение данных и различий с удобной навигацией.



Отображение данных

HEX и символичные представления с указанием смещения.



Цветовое кодирование

Красный для уникальных, цвет исходного для совпадающих.



Навигация

Функции "Найти следующее/предыдущее отличие".

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.10.001 ПЗ

Лист

110



Бизнес-модель и финансы

Проект ориентирован на центры обработки данных и правоохранительные органы. Ожидается быстрая окупаемость.

Целевая аудитория

ЦОД, правоохранительные органы, разработчики, системные администраторы.

Каналы взаимодействия

Сайт-визитка, прямые продажи, партнерство с дистрибьюторами.

Потоки доходов

Продажа лицензий, подписка, услуги внедрения.

Финансовая оценка проекта

Проект демонстрирует положительные финансовые показатели с быстрой окупаемостью.



1286K

Доходы

Ожидаемые доходы на момент окупаемости.

836K

Затраты

Суммарные затраты на проект.

450K

Прибыль

Чистая прибыль проекта.

22

Окупаемость

Срок окупаемости в месяцах.

Заключение и перспективы

Создана программа, способная эффективно работать с файлами большого объема, обеспечивая высокую точность и наглядную визуализацию различий

Перспективы развития: Расширение функционала (отчеты, интеграция с СКВ), дальнейшая оптимизация производительности, адаптация для других платформ.

