

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Операционные системы и системное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

«ПРОГРАММА-РЕВИЗОР»

БГУИР КП 1-40 01 01 18 ПЗ

Студент: гр. 751005 Петрушенко И.А.

Руководитель: Базылев Е.Н.

Минск 2018

СОДЕРЖАНИЕ

Введение.....	4
1 Анализ предметной области и постановка задачи.....	5
1.1 Обзор аналогов.....	5
1.2 Постановка задачи.....	7
2 Разработка программного средства.....	9
2.1 Разработка схемы работы программного средства.....	9
2.2 Интерфейс программного средства.....	9
2.3 Функциональная часть программы.....	13
2.4 Структура программного средства.....	17
3 Тестирование программного средства.....	18
3.1 Проверка работоспособности.....	18
4 Руководство пользователя.....	22
Заключение.....	25
Список использованных источников.....	26
Приложение А. Исходный код программы.....	27

ВВЕДЕНИЕ

Вы куда-то записали файл и не можете его найти? Куда-то исчезло все свободное место на диске? Кто-то из Ваших коллег поработал на машине и пропал нужный файл? Вы запустили новую игру и не знаете, нет ли в ней вируса? Как с этим бороться?

Во всех этих случаях Вам поможет программа-ревизор. Она заметит даже самые незначительные изменения в файловой системе и оповестит Вас. Она поможет Вам поддерживать порядок на дисках, даже заметить некоторые вирусные атаки и не допустить распространения эпидемии вируса на компьютере.

Программа-ревизор – это ревизор дисков, предназначенный для работы на персональных компьютерах под управлением операционных систем, семейства Windows.

Работа программы основана на регулярном отслеживании изменений, происходящих на жестких дисках. По сути, данная программа является системой, позволяющей следить за сохранностью информации на дисках и обнаруживать любые, изменения в файловой системе, а именно изменения файлов, создание, удаление, переименование и перемещение файлов из каталога в каталог.

После первого запуска программы, она определит диски, имеющиеся на вашем компьютере, для них вам нужно будет всего лишь сделать так называемые слепки, которые при каждом следующем сканировании будут обновляться. Результат сканирования выбранного диска покажет вам файлы, которые были изменены, а также удаленные и новые файлы.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ПОСТАНОВКА ЗАДАЧИ

В настоящее время разработано огромное количество антивирусных средств, позволяющих помимо поиска вирусов на компьютере, также сканировать и следить за файловой системой. Но такие программные продукты зачастую являются платные и не доступны любому желающему.

Однако существуют отдельные программы-ревизоры, выполняющие все ту же функцию слежки за файловой системой компьютера. Они будут рассмотрены в данном разделе.

1.1 Обзор аналогов

Первая рассматриваемая программа AVP Inspector представлена на рисунке 1.1 - продукт "Лаборатории Касперского", российского лидера в области разработки антивирусных систем безопасности.

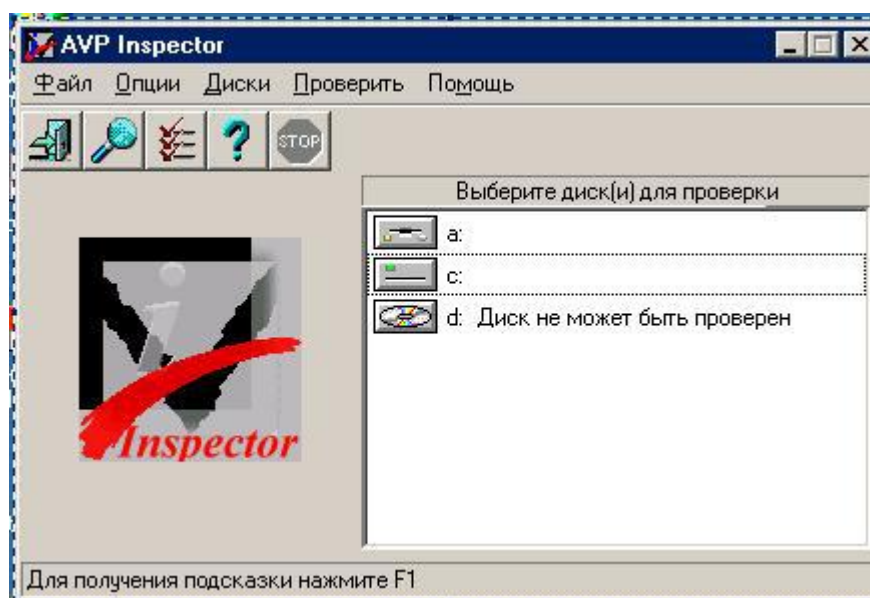


Рисунок 1.1 - Интерфейс AVP Inspector

AVP Inspector является идеальным инструментом отслеживания всех несанкционированных изменений в файлах и загрузочных секторах, находящихся на локальных и сетевых дисках. Принцип работы программы основан на снятии оригинальных "отпечатков" (CRC-сумм) с файлов, каталогов, системного реестра и загрузочных секторов. Эти "отпечатки" сохраняются в базе данных. При следующем запуске ревизор сверяет "отпечатки" с их оригиналами и сообщает пользователю о произошедших изменениях. Таким образом, в отличие от классических антивирусных программ, которые обнаруживают вирусы по их уникальному программному коду, AVP Inspector выявляет их по изменениям, которые они производят в файлах и загрузочных секторах. Если обнаружены подозрительные изменения (похожие на проявления вируса), то AVP Inspector предупредит

Вас о возможности заражения вирусом и предоставит возможность восстановить оригинальное содержимое зараженного объекта.

"Такой способ антивирусной защиты имеет ряд очевидных преимуществ", - комментирует событие Михаил Калиниченко, технический директор "Лаборатории Касперского", - "AVP Inspector не требует постоянных обновлений, поскольку программе не требуется знать вирусы "в лицо". Программа исключительно проста в использовании и предъявляет весьма низкие требования к аппаратной части компьютера. Наконец, AVP Inspector, в отличие от других программ, сможет гарантированно восстановить файлы или загрузочные сектора, которые были повреждены практически любым компьютерным вирусом".

AVP Inspector предоставляет пользователям ряд уникальных особенностей для борьбы с различными видами вирусов. Программа обращается к дискам непосредственно через драйвер дисковой подсистемы IOS (супервизор ввода-вывода). Эта особенность позволяет успешно обнаруживать и нейтрализовывать вирусы-невидимки (Stealth вирусы).

Следовательно, можно выделить явные плюсы данной программы:

- программа проста в использовании;
- работает также в качестве антивируса;
- возможность частичного восстановления поврежденных файлов

Из недостатков можно выделить то, что продукт более не поддерживается, а встраивается в другие антивирусные программы "Лаборатории Касперского", которые являются платными.

Следующая программа также российской разработки - ADinf32. Ее интерфейс представлен на рисунке 1.2:

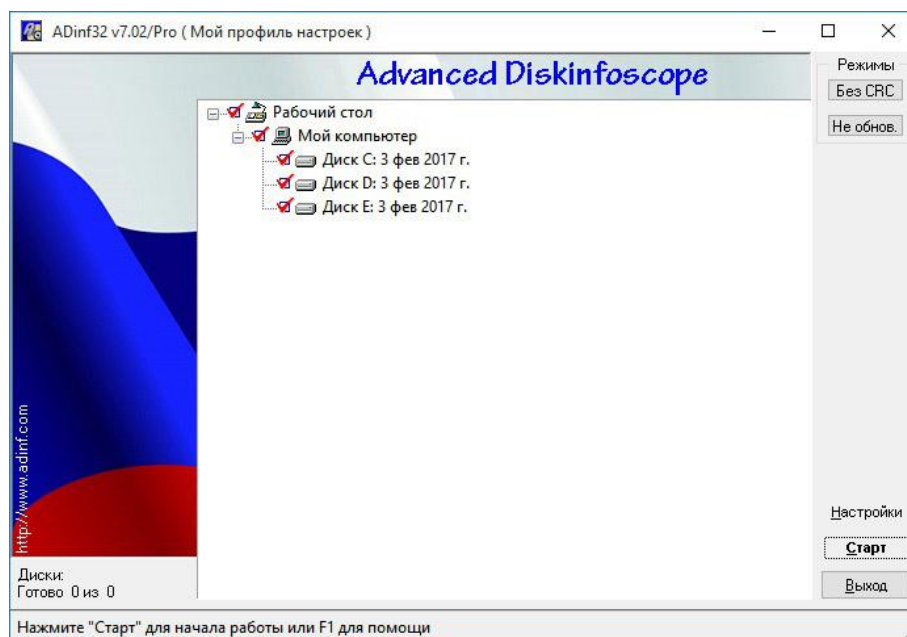


Рисунок 1.2 - ADinf32

ADinf32 - еще одна программа семейства ревизоров, которая имеет, как заявляет разработчик, значительные преимущества в скорости работы перед антивирусными программами.

Главным недостатком антивирусов-сканеров, не позволяющим организовать повседневную проверку дисков, является их невысокая скорость работы. Проверка каждый файл на вирусы, современный сканер проводит эмуляцию выполнения процессорных команд и анализирует файлы на возможное заражение полиморфными вирусами. Все это занимает немало времени. Сканирование на вирусы современных много-гигабайтных дисков может занимать до нескольких часов времени, а использование резидентных сканеров снижает общую производительность системы.

Второй недостаток сканеров - необходимость их регулярного обновления. Только самая последняя версия сканеров со всеми дополнениями базы с описаниями вирусов может обеспечивать адекватную защиту. Сканер, не обновлявшийся две недели, уже устаревает.

Антивирусы-ревизоры, в частности ADinf32, лишены основных недостатков антивирусов-сканеров. Время проверки дисков составляет минуты и программа не требует еженедельного обновления версий и вирусных баз, позволяя при этом обнаруживать, в том числе, новые неизвестные вирусы.

Плюсы данной программы:

- простота использования;
- быстрая скорость работы, по сравнению с антивирусами;
- нет необходимости обновлять вирусную базу.

Минусы:

- условно платная (есть пробный период 30 дней).

1.2 Постановка задачи

Целью данной курсовой работы является создание программы-ревизора, позволяющее пользователю отслеживать изменения на дисках его компьютера.

В программном средстве планируется реализовать ряд функций:

- сбор контрольных сумм файлов на диске;
- быстрый доступ к соответствующей контрольной суммы файла;
- хранение этих контрольных сумм;
- поиск изменений на диске путем сравнения контрольным сумм;
- просмотр результата сканирования.

Для удобного пользования программой необходимо реализовать пользовательский интерфейс, с которым будет взаимодействовать пользователь для выбора нужного ему диска для создания слепок или сканирования, просмотра результата последнего сканирования, а также информации о том, слепки каких дисков имеются в базе даного программного средства.

Разработка будет вестись на языке Си. Среда разработки для этого языка - Microsoft Visual Studio 2017. Использование данной среды разработки и языка дает возможность использовать большое количество функций операционной системы Windows.

Для разработки интерфейса будут задействованы функции WinAPI.

2 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

2.1 Разработка схемы работы программного средства

Исходя из поставленной задачи разработки программы-ревизора, был четко определен функционал программы, а также разработан схематический алгоритм работы программы.

При запуске работы программы она оповещает, сколько дисков на компьютере пользователя было определено, в последствии этот список будет отображаться на главном окне ПС.

Далее пользователю предстоит перед выбором сделать новый отпечаток диска, просканировать диск, либо посмотреть последние результаты работы сканирования. Для этого будут размещены соответствующие кнопки на главном окне программы.

Нажав на кнопку просмотра результатов, пользователь сможет увидеть таблицу с новыми, удаленными и измененными файлами, однако, если сканирования не проводилось, она отображается пустой.

Если пользователь захочет начать сканирование, ему нужно будет выбрать один из определившихся программой дисков на его компьютере, либо выбрать сразу все, после чего программа определит, есть ли отпечаток данного диска или нет, в противном случае, пользователю будет предложено провести слепок выбранного им диска.

После сканирования ему можно будет сразу из окна сканирования посмотреть результаты, либо позже, выбрав соответствующее меню программы.

Можно провести отпечаток сразу же, просто зайдя в соответствующее меню программы.

2.2 Интерфейс программного средства

Проанализировав существующие аналоги программы, с помощью функций WinAPI был разработан пользовательский интерфейс, позволяющий комфортно пользоваться программным средством. Главное окно программы, получило вид, показанный на рисунке 2.1.

На функциональной панели программы находятся три кнопки *Imprint*, *Scan*, *Show Results*, выполняющие соответственно слепок, сканирование дисков и результат сканирования.

Также было сделано главное меню программы, находящееся сверху, в котором есть два пункта: *File* и *About*. Во вкладке *File* доступна вся работа программы, а во вкладке *About* представлена информация о используемом приложении.

За пользователем предстоит выбор, каким способом ему использовать функционал программы: через главное меню или через панель инструментов.

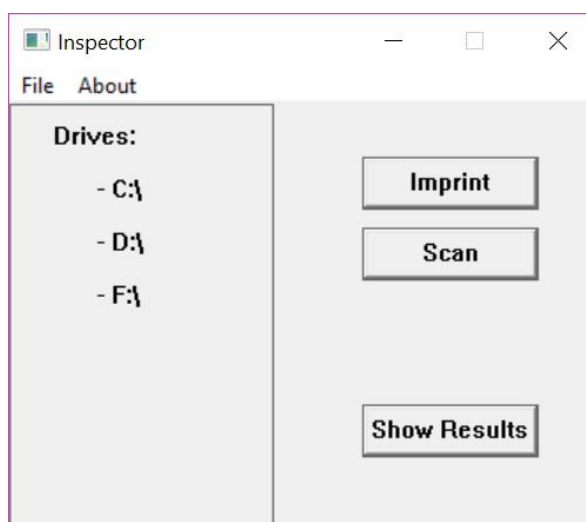


Рисунок 2.1 - Главное окно программы

Перед закрытием программы пользователем будем об этом уведомлен звуковым сигналом, а также сообщением представленном на рисунке 2.2:

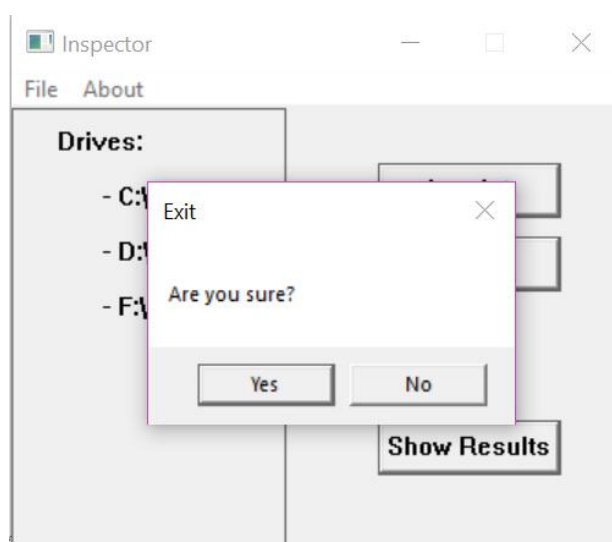


Рисунок 2.2 - Сообщение перед выходом из программы

Нажав на кнопку Imprint или выбрав соответствующее поле в меню, пользователю будет представлено окно для создания отпечатка диска, где можно выбрать диск и начать сканирование, окна представлены на рисунках 2.3 и 2.4:

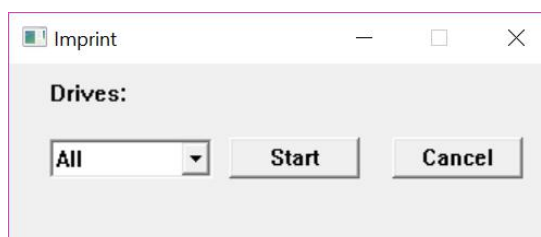


Рисунок 2.3 - Окно Imprint программы

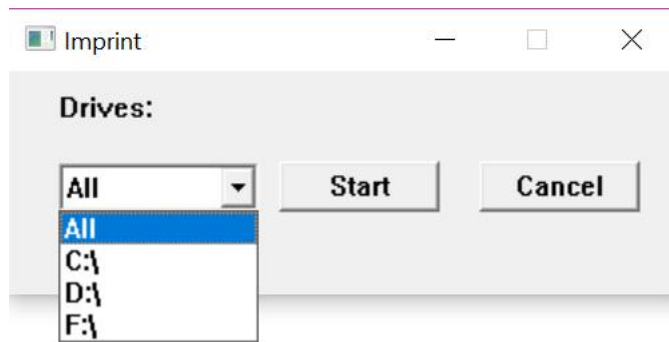


Рисунок 2.4 - Выбор диска для слепок

После нажатия кнопки *Start* начнется сбор контрольных сумм. По окончании создания слепок, пользователь будет уведомлен о количестве отсканированных файлов, результат показан на рисунке 2.5:

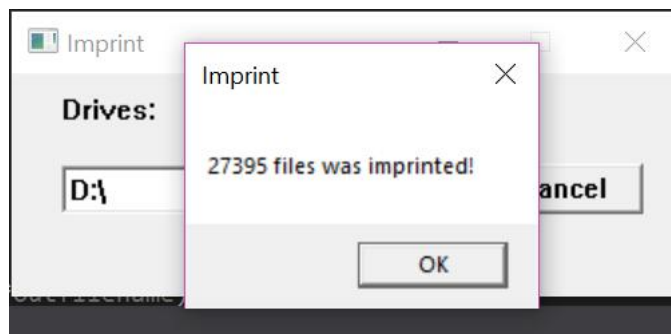


Рисунок 2.5 - Уведомление об отпечатанных файлах

Нажав на кнопку *Scan* на главном окне или выбрав соответствующий пункт меню, пользователь попадает в меню сканирования дисков.

Схема работы такая же, как и при отпечатке диска: выбрать нужный раздел для сканирования и нажать *Start*.

Работу сканирования отображает рисунок 2.6:

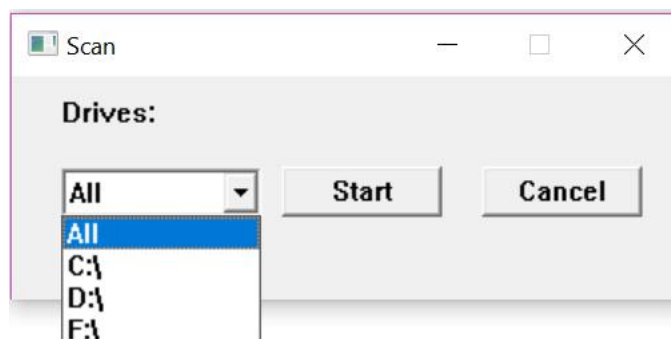


Рисунок 2.6 - Сканирование дисков

После сканирования пользователю будет представлена таблица результатов работы программы, результат представлен на рисунке 2.7.

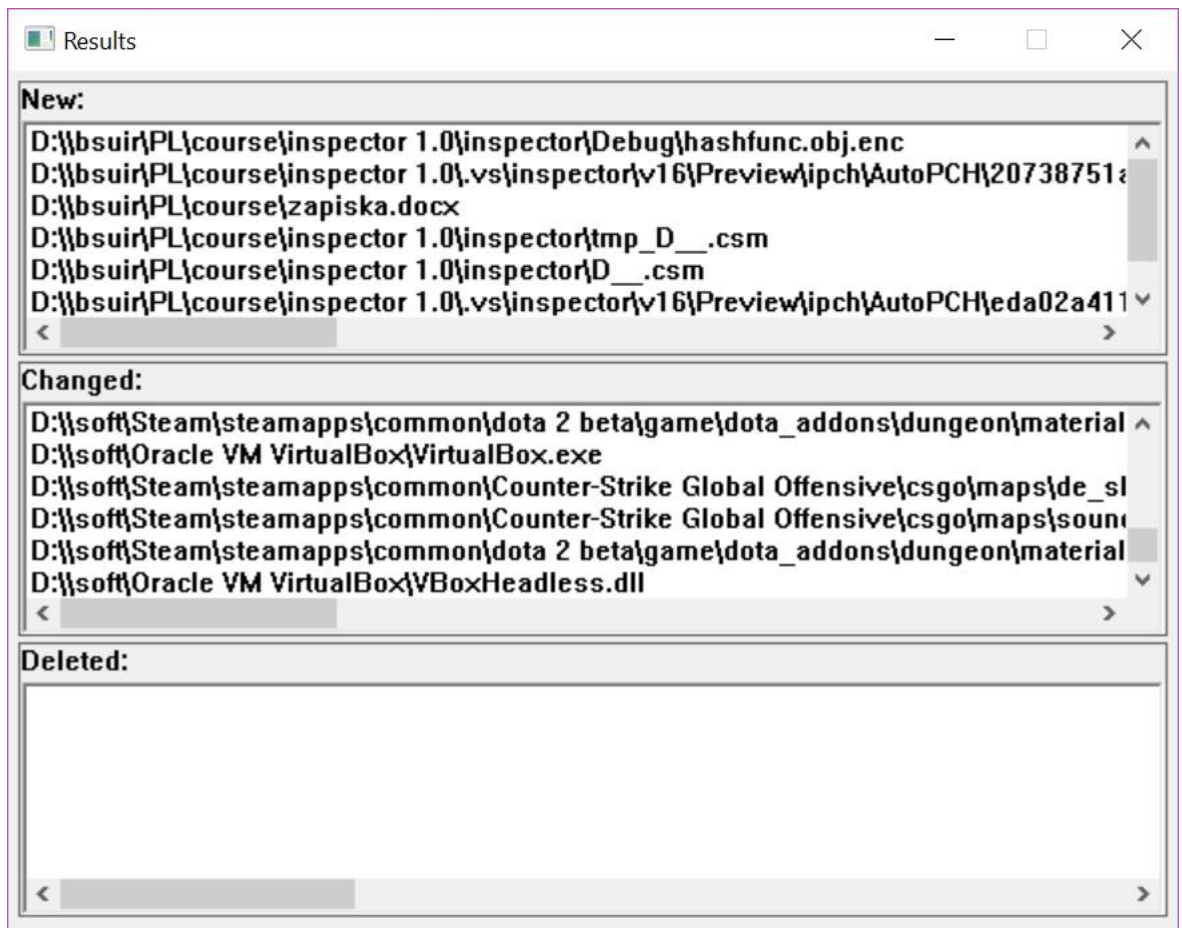


Рисунок 2.7 - Результат работы программы

Все элементы интерфейса разработаны с помощью функций WinAPI. Некоторые из них приведены ниже.

Для создания окна необходимо зарегистрировать нужный класс окна, где указывается имя класса, функцию обработки сообщений и многое другое и создать его, с помощью функции *CreateWindow()*:

```
WNDCLASSW wc = { 0 };
wc.hbrBackground = (HBRUSH)COLOR_WINDOW;
wc.hCursor = LoadCursor(NULL, IDC_ARROW);
wc.hInstance = hInst;
wc.lpszClassName = L"myWindowClass";
wc.lpfnWndProc = WindowProcedure;

if (!RegisterClassW(&wc))
    return -1;

MainWnd = CreateWindowW(wc.lpszClassName, L"Inspector",
    WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_MINIMIZEBOX |
    WS_VISIBLE,
    (ScreenX - WND_WIDTH) / 2,
    (ScreenY - WND_HEIGHT) / 2, WND_WIDTH, WND_HEIGHT, NULL, NULL,
    My_hInst, NULL);
```

Далее, были добавлены все необходимые элементы интерфейса, такие как кнопки, метки и т.п.

Код создания кнопок и меток выглядит следующим образом:

```
//Buttons
CreateWindowW(L"Button", L"Imprint", WS_CHILD | WS_VISIBLE |
    BS_DEFPUSHBUTTON, 200, 30, 100,
    30, hWnd, (HMENU)MENU_IMPRINT, NULL, NULL);

CreateWindowW(L"Button", L"Scan", WS_CHILD | WS_VISIBLE |
    BS_DEFPUSHBUTTON, 200, 70, 100,
    30, hWnd, (HMENU)MENU_SCAN, NULL, NULL);

CreateWindowW(L"Static", L"", WS_CHILD | WS_VISIBLE | WS_BORDER, 0, 0,
    150,
    WND_HEIGHT, hWnd, NULL, NULL, NULL);

CreateWindowW(L"button", L"Show Results", WS_CHILD | WS_VISIBLE |
    BS_DEFPUSHBUTTON, 200, 170, 100,
    30, hWnd, (HMENU)MENU_RESULTS, NULL, NULL);

//Drives:
CreateWindowW(L"Static", L"Drives:", WS_CHILD | WS_VISIBLE, 25, 10,
    60,
    20, hWnd, NULL, NULL, NULL);
```

Самая часто используемая функция является *CreateWindow()*, ее синтаксис приведен на рисунке 2.8.

Syntax

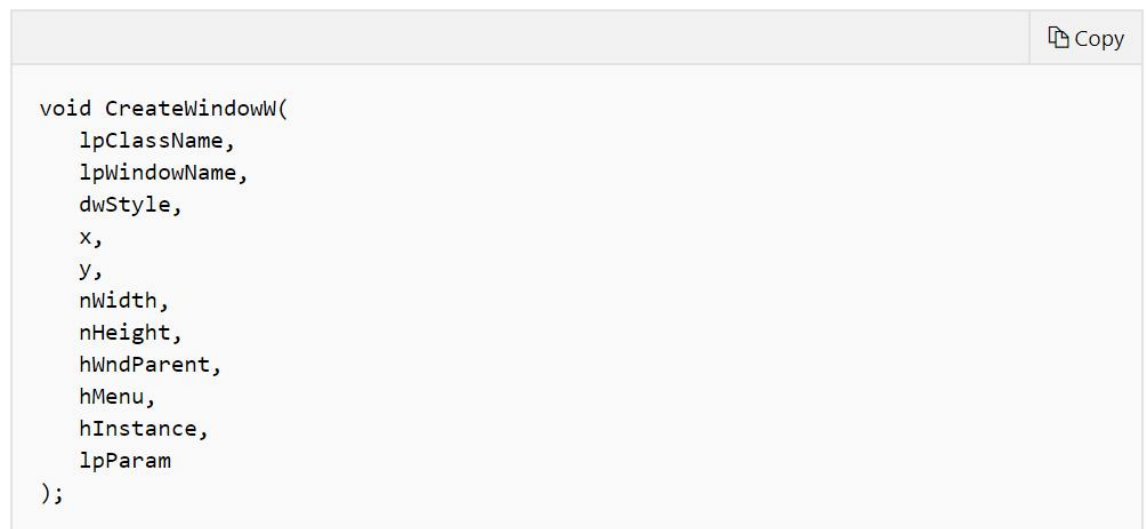


Рисунок 2.8 - Синтаксис функции CreateWindowW

2.3 Функциональная часть программы

Функционал программы реализован в следующих функциях:

- void GetImprint(HWND) - получение слепок дисков пользователя;
- void GetScan(HWND) - сканирование дисков.

Первой функции соответствует алгоритм, укрупненная схема которого представлена на рисунке 2.9.



Рисунок 2.9 - Укрупненная схема алгоритма добавления отпечатков

Для хранения отпечатков каждого файла, используется контрольная сумма `crc32`.

Контрольная сумма — некоторое значение, рассчитанное по набору данных путём применения определённого алгоритма и используемое для проверки целостности данных при их передаче или хранении. Также контрольные суммы могут использоваться для быстрого сравнения двух

наборов данных на неэквивалентность: с большой вероятностью различные наборы данных будут иметь неравные контрольные суммы. Это может быть использовано, например, для обнаружения компьютерных вирусов. Несмотря на своё название, контрольная сумма не обязательно вычисляется путём суммирования.

В общем виде контрольная сумма представляет собой некоторое значение, вычисленное по определённой схеме на основе кодируемого сообщения. Проверочная информация при систематическом кодировании приписывается к передаваемым данным.

Как было сказано, в качестве контрольной суммы используется crc32, которая как раз и предназначена в большей степени для проверки целостности данных.

Контрольные суммы хранятся в хеш-таблице, что является незаменимым средством для быстрого доступа к большому количеству данных.

Чтобы не занимать много места в оперативной памяти, данные о всех файлах из таблицы сразу же сохраняются в файл, а сама таблица удаляется.

Для функции GetScan укрупненная схема алгоритма отображена на рисунке 2.11.

Из нее видно, что перед началом сканирования, проверяется, есть ли отпечаток выбранного диска или нет, если отпечатка нет, об этом информируется пользователь и ему предлагается сделать отпечаток, если он согласится, начнется процедура создания отпечатка диска, в случае отказа - сканирование не будет произведено. Как только у диска появится отпечаток, можно проводить сканирование.

При сканировании, программа определяет какие файлы были изменены, удалены, а также находит новые файлы на диске.

Все полученные данные можно будет посмотреть в таблице результата.

После процедуры сканирования, пользователь будет оповещен об этом и может просмотреть результат сразу же, нажав соответствующую кнопку(рис. 2.10).

Для потребления меньшей оперативной памяти компьютера, как и в случае создания отпечатка, данные хеш-таблицы сохраняются в файл, тем самым отпечаток обновляется, а сама хеш-таблица удаляется.

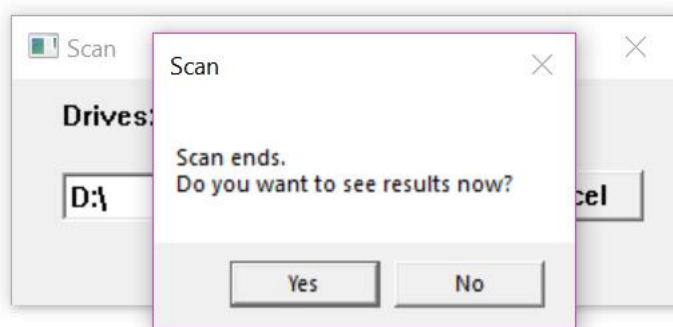


Рисунок 2.10 - Оповещение пользователя о результате сканирования

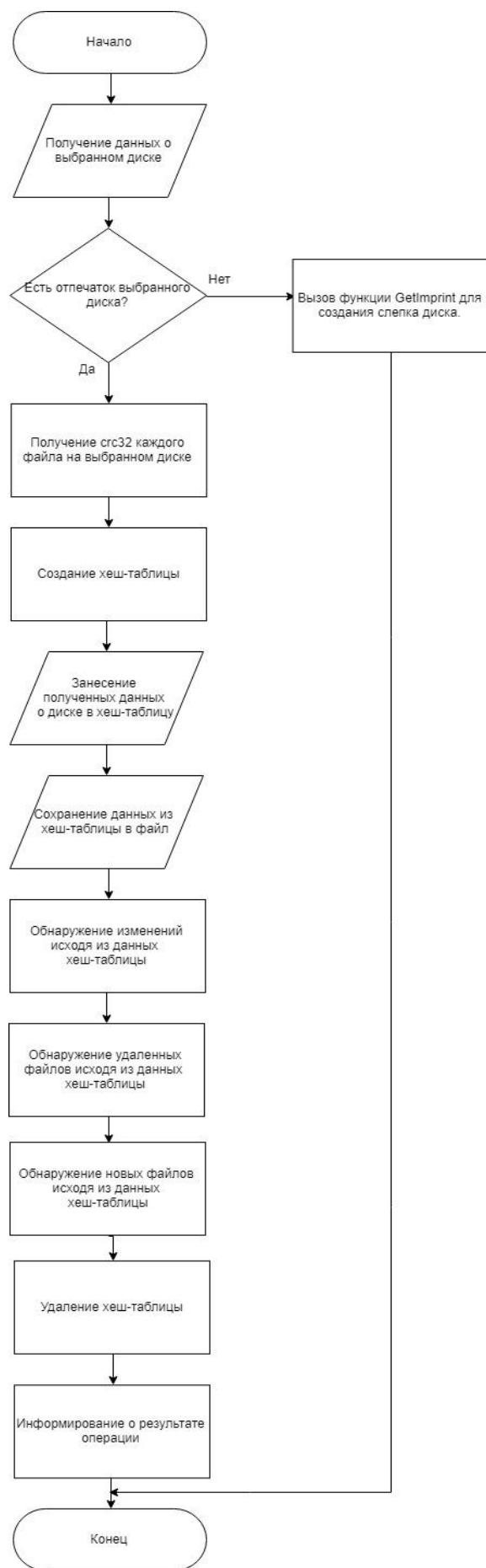


Рисунок 2.11 - Сканирование диска

2.4 Структура программного средства

В итоге программа реализована в пяти модулях:

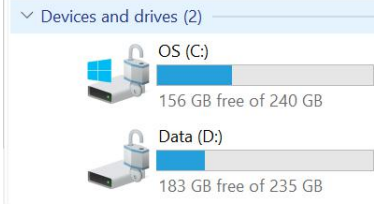
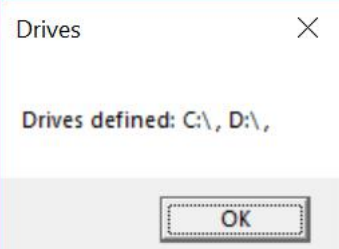
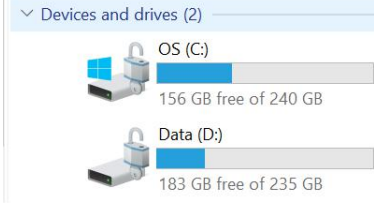
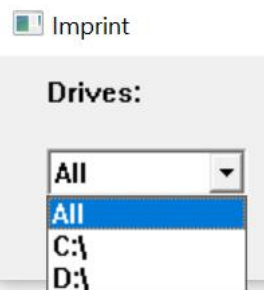
- main.c - модуль, в котором реализована интерфейсная часть программы, а также обращение к функциональной части других модулей;
- filehash.c - реализация функций работы с файлами;
- hashfunc.c - функции, которые высчитывают контрольные суммы файлов
- dirview.c - модуль с функциями для прохода по всем файлам диска;
- hashtable.c - основная работа с хеш-таблицей, в этом модуле и происходит определение, какие файлы на диске подверглись изменению, какие файлы появились и т.п.

3 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

3.1 Проверка работоспособности

Для начала был проведен тест на правильность определения дисков на компьютере. Ниже представлена таблица с соответствующими тестами.

Таблица 3.1 - Тесты определения дисков

№	Тестируемая функция	Ожидаемый результат	Полученный результат
1	Определение дисков на компьютере		
2	Определение дисков на компьютере		

Тесты показывают, что диски определяются верно.

Далее необходим тест на правильное функционирование возможности программы определять удаленные, измененные и новые файлы.

Для этого был отсканирован диск D:\, создана папка *tests* и там создан новый файл *Delete.txt*, *Changed.txt* с некоторой информацией на них, это представлено на рисунка 3.1 и 3.2.

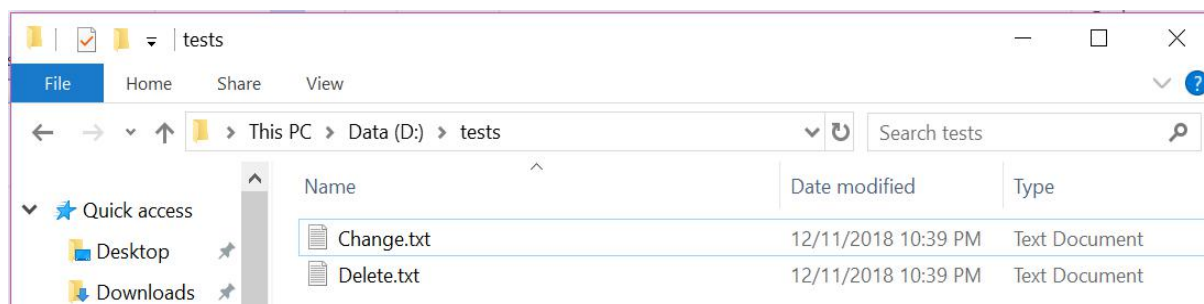


Рисунок 3.1 - Созданные файлы



Рисунок 3.2 - Информация в файле Change.txt

Результат теста на определение новых файлов показан в таблице 3.2:

Таблица 3.2 - Тест определения новых файлов

№	Тестируемая функция	Ожидаемый результат	Полученный результат
3	Определение новых файлов на диске	Добавлены файлы <i>Delete.txt</i> , <i>Changed.txt</i>	Рисунок 3.3

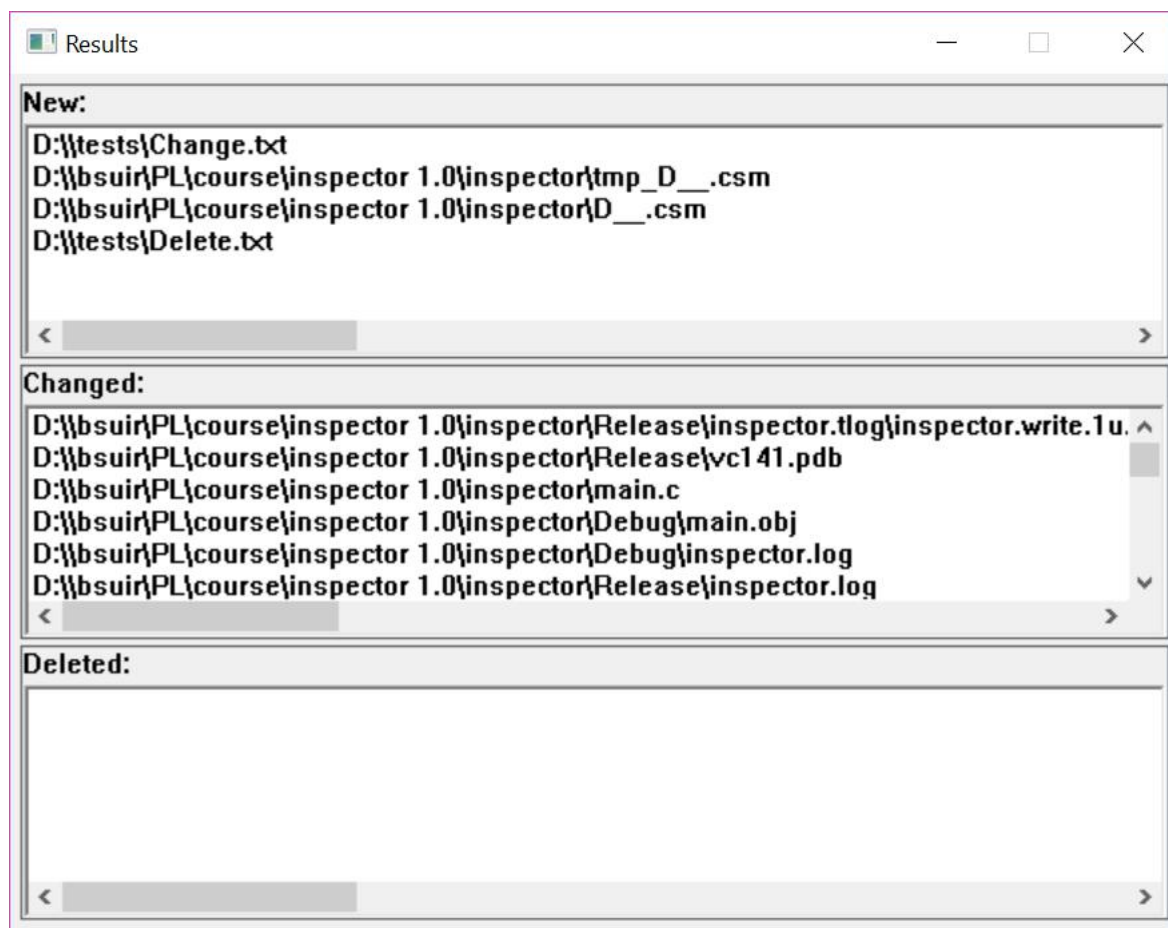


Рисунок 3.3 - Результат теста №3

Тест был пройден успешно. Кроме файлов *Delete.txt*, *Changed.txt* показаны другие файлы. Это объясняется работой среды Visual Studio 2017, в которой тестировалась программы.

Теперь изменив файл *Change.txt* и удалив файл *Delete.txt*, был проведен очередной тест. Результат показан на рисках 3.4 и 3.5

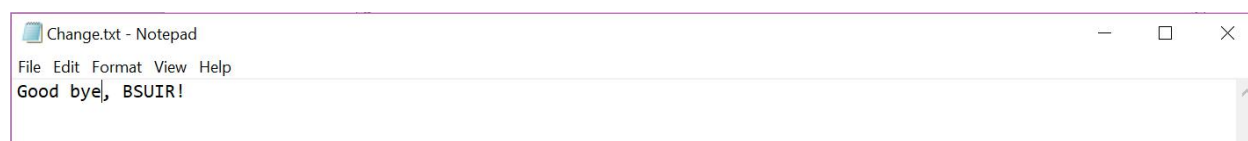


Рисунок 3.4 - Изменение файла Change.txt

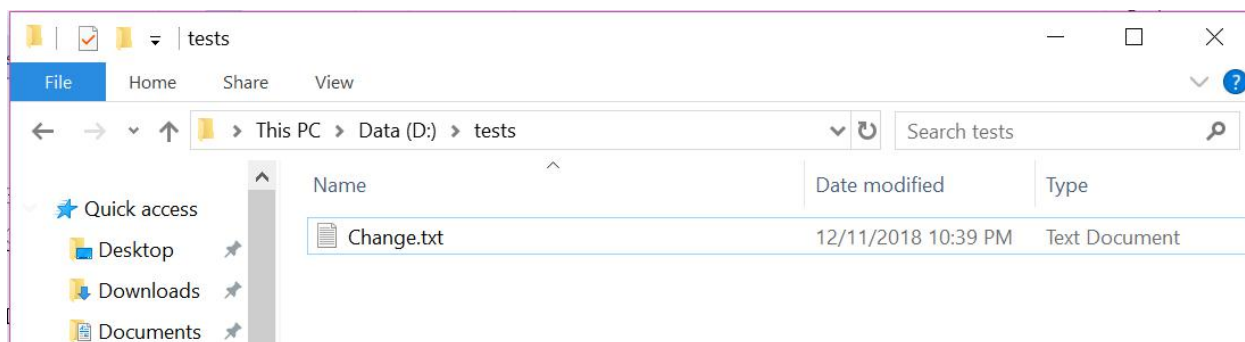


Рисунок 3.5 - Удаление файла Delete.txt

Результат сканирования показан в таблице 3.3:

Таблица 3.3 - Второе сканирование программы

№	Тестируемая функция	Ожидаемый результат	Полученный результат
3	Определение измененных и удаленных файлов	Файл Delete.txt удален, Файл Change.txt изменен	Рисунок 3.4

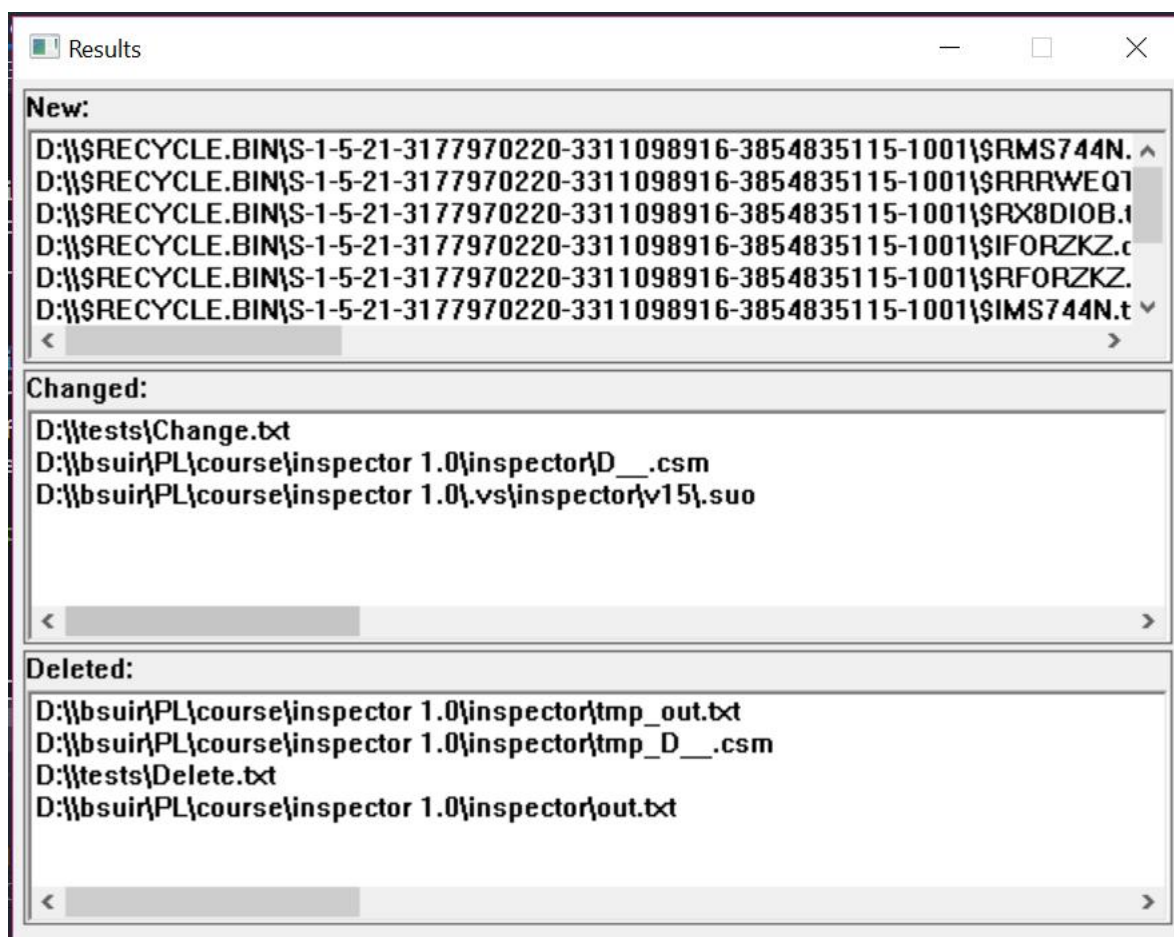


Рисунок 3.4 - Результат сканирования

Из проведенных тестов видно, что программа находит новые файлы, определяет удаленные и измененные файлы на выбранном пользователем диске.

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

При первом запуске программа оповещает о найденных на компьютере дисках, результат показан на рисунке 4.1, после нажатия на «ОК», появляется главное окно программы, на котором находятся функциональные кнопки «Imprint», «Scan», «Show Results».

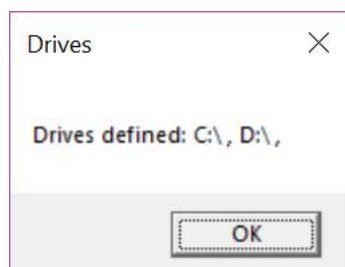


Рисунок 4.1 - Найденные диска на компьютере

По нажатию на «Imprint» (или *File->Imprint*) открывается окно создания отпечатка диска, это показано на рисунке 4.2, где выбирается нужный диск, как на рисунке 4.3 и, нажав на «Start», начинается сбор контрольных сумм всех файлов на диске. Для отмены, до начала сбора нужно нажать «Cancel» - возвращается главное окно программы.

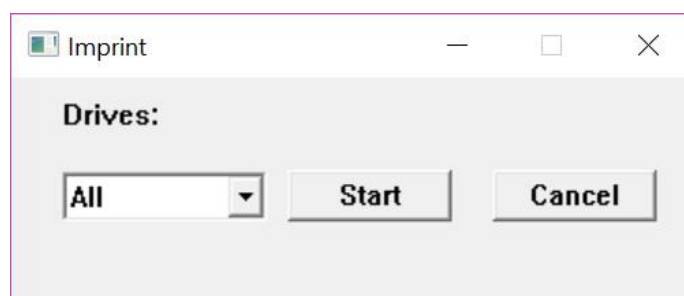


Рисунок 4.2 - Окно «Imprint»

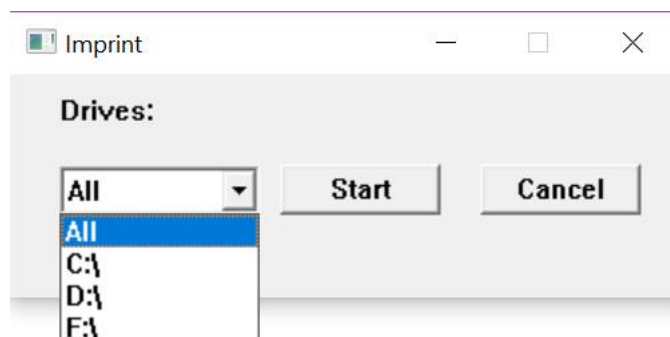


Рисунок 4.3 - Выбор диска

Для сканирования на главном окне программы нажимается «Scan», после чего открывается окно сканирования диска, где выбирается диск, и кнопка «Start» запустит сканирование.

Окно сканирования отображено на рисунке 4.4:

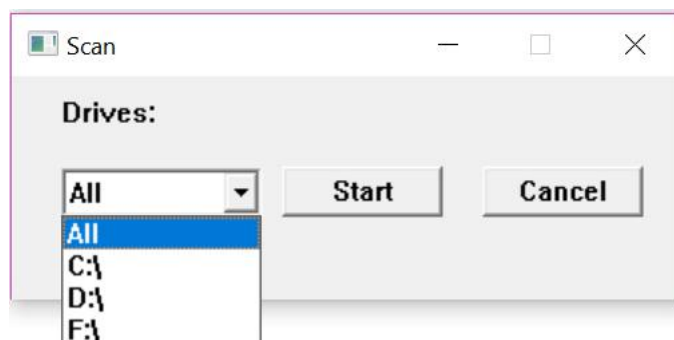


Рисунок 4.4 - Окно сканирования

После операций сканирования и создания отпечатков, пользователь уведомляется об окончании работы. После этого, он может вернуться в главное меню или посмотреть результат сканирования в случае сканирования. Окно результата сканирования выглядит следующим образом, как показано на рисунке 4.5:

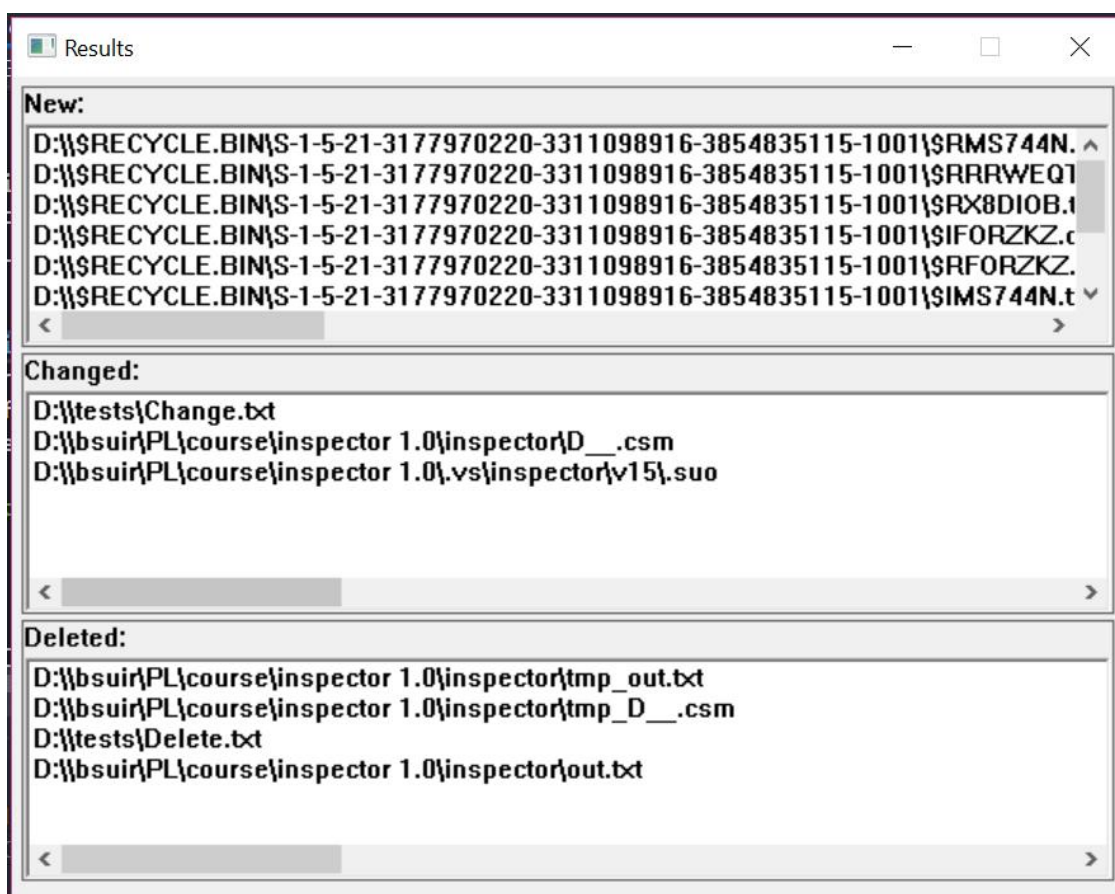


Рисунок 4.5 - Окно результата

Нажатие на крестик в правом верхнем углу (или *File->Exit*) откроет предупреждение о закрытии программы, результат на 4.6, чтобы закрыть программу, нужно нажать на «Yes», для возврата в главное меню - «No».

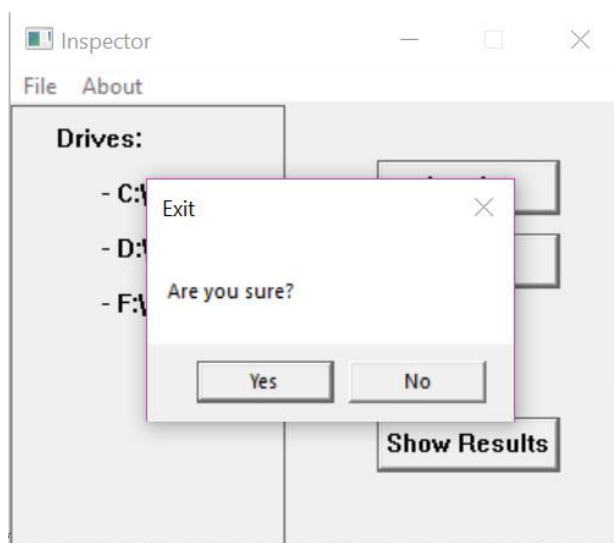


Рисунок 4.6 - Выход из программы

ЗАКЛЮЧЕНИЕ

Целью курсового проекта являлось программное средство «Программа-ревизор», которое облегчает мониторинг за файловой системой компьютера.

В ходе выполнения проекта были закреплены знания языка Си и работы с ним в среде разработки Visual Studio 2017.

Для разработки данного приложения требовалось решить множество задач, вследствие чего было проанализировано большое количество программ со схожей тематикой и сформированы требования для приложения такого типа.

Был разработан и реализован интерфейс программного средства с использованием функций WinAPI и функционал работы программы.

Убедившись в корректности работы программы после пройденных тестов, можно сказать, что результат курсового проекта «Программа-ревизор» полностью соответствует всем требованиям, поставленным в начале разработки программы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Microsoft Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.microsoft.com/ru-ru/visualstudio/ide/?view=vs-2017>
- [2] MSDN [Электронный ресурс]. - Электронные данные. - Режим доступа: <https://msdn.microsoft.com/en-us/dn308572.aspx>
- [3] StackOverflow [Электронный ресурс]. - Электронные данные. - Режим доступа: <https://stackoverflow.com/>
- [4] Дж. Рихтер. Создание эффективных WIN32-приложений с учетом специфики 64-разрядной версии Windows. - Москва, 2008. - 743с.
- [5] Х. Дейтел, П. Дейтел. Как программировать на С. - Питер, 2008. - 1004с.
- [6] Wikipedia [Электронный ресурс]. - Электронные данные. - Режим доступа: <https://ru.wikipedia.org>
- [7] Уилсон, С. Принципы проектирования и разработки программного обеспечения, учеб. курс. – СПб. : Русская Редакция, 2003 – 570 с.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
////////////////////////////////////
//
// hashtable.c - функции работы с хеш-таблицей
//
////////////////////////////////////

#include "hashtable.h"
#include "hashfunc.h"
#include "filehash.h"
#include <stdlib.h>
#include <string.h>
#include "dirview.h"

/* http://en.wikipedia.org/wiki/Jenkins_hash_function */
uint32_t _htable_hash(const char *key, const size_t key_len) {
    uint32_t hash, i;
    for (hash = i = 0; i < key_len; ++i) {
        hash += key[i];
        hash += (hash << 10);
        hash ^= (hash >> 6);
    }
    hash += (hash << 3);
    hash ^= (hash >> 11);
    hash += (hash << 15);
    return hash % (HASHTABLE_MIN_SIZE);
}
/*
Инициализация хеш-таблицы
*/
void ht_init(struct ht_item_t **tbl)
{
    //tbl = malloc(sizeof(ht_item_t)*HASHTABLE_MIN_SIZE);
    for (int i = 0; i < HASHTABLE_MIN_SIZE; i++) {
        tbl[i] = NULL;
    }
}
/*
Добавление в хеш таблицу
*/
void ht_add(struct ht_item_t **tbl, char *fname, uint32_t crc32) {

    ht_item_t *item;

    uint32_t index = _htable_hash(fname, strlen(fname));

    item = malloc(sizeof(ht_item_t));
    if (item) {
        strcpy_s((item->fname), PATHMAXSIZE, fname);
        item->crc32 = crc32;
        item->pnext = tbl[index];
        //item->pnext = tbl + index;
        /*(tbl + index) = item;
        tbl[index] = item;
    }
}
/*
Получение СПИСКА найденных элементов из хеш таблицы
*/
ht_item_t *ht_get(struct ht_item_t **tbl, char *fname)
{
    uint32_t index = _htable_hash(fname, strlen(fname));
```

```

    ht_item_t *ht_item = tbl[index];

    ht_item_t *res = NULL;

    ht_item_t *tmp_res;

    while (ht_item) {
        if (strcmp(fname, ht_item->fname) == 0) {
            tmp_res = malloc(sizeof(ht_item_t));
            if (tmp_res) {
                tmp_res->crc32 = ht_item->crc32;
                //tmp_res->fname = ht_item->fname;
                strcpy_s(tmp_res->fname, PATHMAXSIZE, ht_item->fname);
                tmp_res->pnext = res;
                res = tmp_res;
            }
            else
                return NULL;
        }
        ht_item = ht_item->pnext;
    }
    return res;
}

/*
Удаление из хеш таблицы
*/
void ht_delete(struct ht_item_t **tbl, char *fname, uint32_t crc32)
{
    uint32_t index = _htable_hash(fname, strlen(fname));

    ht_item_t *tmp_ptr, *to_free;

    tmp_ptr = tbl[index];

    if (tmp_ptr && strcmp(tmp_ptr->fname, fname) == 0 && tmp_ptr->crc32 ==
crc32) {
        tbl[index] = tmp_ptr->pnext;
        free(tmp_ptr);
        return;
    }

    while (tmp_ptr && tmp_ptr->pnext) {
        if (strcmp(tmp_ptr->pnext->fname, fname) == 0 && tmp_ptr->pnext-
>crc32 == crc32) {
            to_free = tmp_ptr->pnext;
            tmp_ptr->pnext = tmp_ptr->pnext->pnext;
            free(to_free);
        }
        tmp_ptr = tmp_ptr->pnext;
    }
}

/*
Очистка хеш таблицы
*/
void ht_clear(struct ht_item_t **tbl)
{
    ht_init(tbl);
}

void ht_destroy(struct ht_item_t **tbl)
{
    ht_item_t *ht_item, *prev_ht_item;

```

```

        for (int i = 0; i < HASHTABLE_MIN_SIZE; i++) {
            ht_item = tbl[i];
            while (ht_item) {
                prev_ht_item = ht_item;
                ht_item = ht_item->pnext;
                ht_delete(tbl, prev_ht_item->fname, prev_ht_item->crc32);
            }
        }
        free(tbl);
    }

    //Поиск изменений на диске
    uint32_t _ht_chngs(ht_item_t **newf, ht_item_t **del, ht_item_t **chn, const char
*fname, const char *dir)
    {
        hashtable_t *hash_table = _ht_init();

        ht_item_t *ht_item = NULL, *changed = NULL, *deleted = NULL, *pnew = NULL,
*tmpPtr = NULL;

        FILE *f;
        char new_fname[PATHMAXSIZE] = "tmp_";
        strcat(new_fname, fname);

        errno_t err = fopen_s(&f, fname, "r");

        uint32_t crc = 0; //crc32 прочитанного файла
        char filename[PATHMAXSIZE]; //имя прочитанного файла

        if (!err && f) {
            ListDirectoryContents(dir, hash_table->ht_items);
            fhash_save(hash_table->ht_items, new_fname);
            while (!feof(f)) {
                fgets(filename, PATHMAXSIZE, f);
                if (filename)
                    filename[strlen(filename) - 1] = '\0';
                else
                    return -1;
                fscanf_s(f, "%u\n", &crc);
                ht_item = ht_get(hash_table->ht_items, filename);
                // см. КОММЕНТ НИЖЕ
                if (!ht_item) {
                    tmpPtr = malloc(sizeof(ht_item_t));
                    tmpPtr->crc32 = crc;
                    strcpy_s(tmpPtr->fname, PATHMAXSIZE, filename);
                    tmpPtr->pnext = deleted;
                    deleted = tmpPtr;
                }
                else {
                    while (ht_item) {
                        if (strcmp(ht_item->fname, filename) == 0) {
                            if (ht_item->crc32 != crc) {
                                tmpPtr = malloc(sizeof(ht_item_t));
                                tmpPtr->crc32 = ht_item->crc32;
                                strcpy_s(tmpPtr->fname, PATHMAXSIZE,
ht_item->fname);

                                tmpPtr->pnext = changed;
                                changed = tmpPtr;
                                ht_delete(hash_table->ht_items,
tmpPtr->fname, tmpPtr->crc32);

                                }
                                else//удаление из хеш-таблицы
                                    ht_delete(hash_table->ht_items,
filename, crc);
                            }
                        }
                    }
                }
            }
        }
    }

```

```

        }
        ht_item = ht_item->pnext;
    }
}
fclose(f);
for (uint32_t i = 0; i < HASHTABLE_MIN_SIZE; i++) {
    ht_item = hash_table->ht_items[i];
    while (ht_item) {
        tmpPtr = malloc(sizeof(ht_item_t));
        tmpPtr->crc32 = ht_item->crc32;
        strcpy_s(tmpPtr->fname, PATHMAXSIZE, ht_item->fname);
        tmpPtr->pnext = pnew;
        pnew = tmpPtr;
        ht_item = ht_item->pnext;
        ht_delete(hash_table->ht_items, tmpPtr->fname, tmpPtr->
>crc32);
    }
}
fhash_load(hash_table->ht_items, new_fname);
fhash_save(hash_table->ht_items, fname);
DeleteFileA(new_fname);

ht_destroy(hash_table->ht_items);
free(hash_table);
}
else
    return -1;

*del = deleted;
*chn = changed;
*newf = pnew;

return 0;
}
/*
Инициализация хеш-таблицы
*/
htable_t *_ht_init(void)
{
    ht_item_t **_ht_items;
    htable_t *tbl = malloc(sizeof(htable_t));

    if (!tbl) {
        return NULL;
    }

    tbl->dir = "\\0";
    tbl->nitems = 0;
    tbl->size = HASHTABLE_MIN_SIZE;

    _ht_items = malloc(sizeof(ht_item_t*) * tbl->size);

    ht_init(_ht_items);

    if (!_ht_items) {
        free(tbl);
        return NULL;
    }

    memset(_ht_items, 0, sizeof(ht_item_t*) * tbl->size);
    tbl->ht_items = _ht_items;

    return tbl;
}

```

```

////////////////////////////////////
//
//  hashfunc.c - реализация функции контрольной суммы
//
////////////////////////////////////

#include "hashfunc.h"
#include <stdint.h>

/*
  Name   : CRC-32
  Poly   : 0x04C11DB7       $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11}$ 
                                    $+ x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ 

  Init   : 0xFFFFFFFF
  Revert : true
  XorOut : 0xFFFFFFFF
  Check  : 0xCBF43926 ("123456789")
  MaxLen : 268 435 455 байт (2 147 483 647 бит) - обнаружение
            одинарных, двойных, пакетных и всех нечетных ошибок
*/

const uint_least32_t Crc32Table[256] = {
    0x00000000, 0x77073096, 0xEE0E612C, 0x990951BA,
    0x076DC419, 0x706AF48F, 0xE963A535, 0x9E6495A3,
    0x0EDB8832, 0x79DCB8A4, 0xE0D5E91E, 0x97D2D988,
    0x09B64C2B, 0x7EB17CBD, 0xE7B82D07, 0x90BF1D91,
    0x1DB71064, 0x6AB020F2, 0xF3B97148, 0x84BE41DE,
    0x1ADAD47D, 0x6DDDE4EB, 0xF4D4B551, 0x83D385C7,
    0x136C9856, 0x646BA8C0, 0xFD62F97A, 0x8A65C9EC,
    0x14015C4F, 0x63066CD9, 0xFA0F3D63, 0x8D080DF5,
    0x3B6E20C8, 0x4C69105E, 0xD56041E4, 0xA2677172,
    0x3C03E4D1, 0x4B00D447, 0xD20085FD, 0xA50AB56B,
    0x35B5A8FA, 0x42B2986C, 0xDBBBC9D6, 0xACBCF940,
    0x32D86CE3, 0x45DF5C75, 0xDCD60DCF, 0xABD13D59,
    0x26D930AC, 0x51DE003A, 0xC8D75180, 0xBFDD06116,
    0x21B4F4B5, 0x56B3C423, 0xCFBA9599, 0xB8BDA50F,
    0x2802B89E, 0x5F058808, 0xC60CD9B2, 0xB10BE924,
    0x2F6F7C87, 0x58684C11, 0xC1611DAB, 0xB6662D3D,
    0x76DC4190, 0x01DB7106, 0x98D220BC, 0xEFD5102A,
    0x71B18589, 0x06B6B51F, 0x9FBBE4A5, 0xE8B8D433,
    0x7807C9A2, 0x0F00F934, 0x9609A88E, 0xE10E9818,
    0x7F6A0DBB, 0x086D3D2D, 0x91646C97, 0xE6635C01,
    0x6B6B51F4, 0x1C6C6162, 0x856530D8, 0xF262004E,
    0x6C0695ED, 0x1B01A57B, 0x8208F4C1, 0xF50FC457,
    0x65B0D9C6, 0x12B7E950, 0x8BBEB8EA, 0xFCB9887C,
    0x62DD1DDF, 0x15DA2D49, 0x8CD37CF3, 0xFBD44C65,
    0x4DB26158, 0x3AB551CE, 0xA3BC0074, 0xD4BB30E2,
    0x4ADFA541, 0x3DD895D7, 0xA4D1C46D, 0xD3D6F4FB,
    0x4369E96A, 0x346ED9FC, 0xAD678846, 0xDA60B8D0,
    0x44042D73, 0x33031DE5, 0xAA0A4C5F, 0xDD0D7CC9,
    0x5005713C, 0x270241AA, 0xBE0B1010, 0xC90C2086,
    0x5768B525, 0x206F85B3, 0xB966D409, 0xCE61E49F,
    0x5EDEF90E, 0x29D9C998, 0xB0D00982, 0xC7D77A8B4,
    0x59B33D17, 0x2EB40D81, 0xB7BD5C3B, 0xC0BA6CAD,
    0xEDB88320, 0x9ABFB3B6, 0x03B6E20C, 0x74B1D29A,
    0xEAD54739, 0x9DD277AF, 0x04DB2615, 0x73DC1683,
    0xE3630B12, 0x94643B84, 0x0D6D6A3E, 0x7A6A5AA8,
    0xE40ECF0B, 0x9309FF9D, 0x0A00AE27, 0x7D079EB1,
    0xF00F9344, 0x8708A3D2, 0x1E01F268, 0x6906C2FE,
    0xF762575D, 0x806567CB, 0x196C3671, 0x6E6B06E7,
    0xFED41B76, 0x89D32BE0, 0x10DA7A5A, 0x67DD4ACC,
    0xF9B9DF6F, 0x8EBEEFF9, 0x17B7BE43, 0x60B08ED5,
    0xD6D6A3E8, 0xA1D1937E, 0x38D8C2C4, 0x4FDDFF252,
    0xD1BB67F1, 0xA6BC5767, 0x3FB506DD, 0x48B2364B,
    0xD80D2BDA, 0xAF0A1B4C, 0x36034AF6, 0x41047A60,

```

```

0xDF60EFC3, 0xA867DF55, 0x316E8EEF, 0x4669BE79,
0xCB61B38C, 0xBC66831A, 0x256FD2A0, 0x5268E236,
0xCC0C7795, 0xBB0B4703, 0x220216B9, 0x5505262F,
0xC5BA3BBE, 0xB2BD0B28, 0x2BB45A92, 0x5CB36A04,
0xC2D7FFA7, 0xB5D0CF31, 0x2CD99E8B, 0x5BDEAE1D,
0x9B64C2B0, 0xEC63F226, 0x756AA39C, 0x026D930A,
0x9C0906A9, 0xEB0E363F, 0x72076785, 0x05005713,
0x95BF4A82, 0xE2B87A14, 0x7BB12BAE, 0x0CB61B38,
0x92D28E9B, 0xE5D5BE0D, 0x7CDECB7, 0x0BDBDF21,
0x86D3D2D4, 0xF1D4E242, 0x68DD3F8, 0x1FDA836E,
0x81BE16CD, 0xF6B9265B, 0x6FB077E1, 0x18B74777,
0x88085AE6, 0xFF0F6A70, 0x66063BCA, 0x11010B5C,
0x8F659EFF, 0xF862AE69, 0x616BFFD3, 0x166CCF45,
0xA00AE278, 0xD70DD2EE, 0x4E048354, 0x3903B3C2,
0xA7672661, 0xD06016F7, 0x4969474D, 0x3E6E77DB,
0xAED16A4A, 0xD9D65ADC, 0x40DF0B66, 0x37D83BF0,
0xA9BCAE53, 0xDEBB9EC5, 0x47B2CF7F, 0x30B5FFE9,
0xBDBDF21C, 0xCABAC28A, 0x53B39330, 0x24B4A3A6,
0xBAD03605, 0xCDD70693, 0x54DE5729, 0x23D967BF,
0xB3667A2E, 0xC4614AB8, 0x5D681B02, 0x2A6F2B94,
0xB40BBE37, 0xC30C8EA1, 0x5A05DF1B, 0x2D02EF8D
};

uint_least32_t Crc32(const unsigned char * buf, size_t len)
{
    uint_least32_t crc = 0xFFFFFFFF;
    while (len--)
        crc = (crc >> 8) ^ Crc32Table[(crc ^ *buf++) & 0xFF];
    //return crc ^ 0xFFFFFFFFFUL;
    // ИНВЕРТИРОВАТЬ ПОЛУЧЕННОЕ ХЕШ-ЗНАЧЕНИЕ
    return crc;
}

////////////////////////////////////
//
// filehash.c - реализация функций для работы с файлами
//
////////////////////////////////////

#include "filehash.h"
#include "dirview.h"

/*
Получение контрольной суммы из файла
Вычисляется CRC32 с полиномом 0x8005
См. описание Crc32()
*/
uint_least32_t fgetcrc32(char *filename)
{
    FILE *f;
    uint_least32_t crc = 0xFFFFFFFFFUL;
    errno_t err;
    err = fopen_s(&f, filename, "rb");
    if (err == 0 && f != NULL) {
        while (!feof(f)) {
            int result = fread(buffer, 1, BUFFSIZE, f);
            if (result) {
                crc = Crc32(buffer, result, crc);
            }
        }
        fclose(f);
        //int size = (BUFFSIZE * sizeof(byte_t));
        //ZeroMemory(buffer, size);
    }
}

```

```

    }

    // Значение инвертируется в конце для возможности в цикле
    // вычислять хеш-функцию частями
    return crc ^ 0xFFFFFFFFFUL;
}
/*
Сохранение данных о файлах хеш таблицы в файл
Возвращает количество обработанных файлов
*/
int fhash_save(struct ht_item_t **tbl, const char *outfilename) {
    int res = 0;
    FILE *f;
    errno_t err = fopen_s(&f, outfilename, "w");
    ht_item_t *tmp_ptr;

    if (err == 0 && f != NULL) {
        for (uint32_t i = 0; i < HASHTABLE_MIN_SIZE; i++) {
            tmp_ptr = *(tbl + i);
            while (tmp_ptr) {
                //fputs(tmp_ptr->fname, f);
                //fprintf_s(f, "\n%u\n", tmp_ptr->crc32);
                fprintf(f, "%s\n%u\n", (tmp_ptr->fname), (tmp_ptr->
>crc32));

                tmp_ptr = tmp_ptr->pnext;
                res += 1;
            }
        }
        fclose(f);
    }
    return res;
}
/*
Загрузка данных о файлах в хеш таблицу
*/
int fhash_load(struct ht_item_t **tbl, const char *infilename) {

    int res = 0;
    FILE *f;
    uint32_t crc = 0; //crc32 прочитанного файла
    char filename[PATHMAXSIZE]; //имя прочитанного файла
    errno_t err = fopen_s(&f, infilename, "r");
    char symb;

    if (!err && f) {
        while (!feof(f)) {
            fgets(filename, PATHMAXSIZE, f);
            if (filename)
                filename[strlen(filename) - 1] = '\0';
            fscanf_s(f, "%u\n", &crc);
            //fscanf_s(f, "%s\n%u\n", filename, PATHMAXSIZE, &crc);
            ht_add(tbl, filename, crc);
            res += 1;
        }
        fclose(f);
    }
    return res;
}

////////////////////////////////////
//
//  dirview.c - функции поиска файлов по папкам
//
////////////////////////////////////

```



```

#include "dirview.h"
#include "filehash.h"
#include <Windows.h>
#include <stdbool.h>
#include <string.h>

int fname_to_str(char *fname)
{
    int res = 0;
    for (int i = 0; i < strlen(fname); i++) {
        if (*(fname + i) == '\\\\' || *(fname + i) == ':') {
            *(fname + i) = '_';
            res++;
        }
    }
    return res;
}

// Поиск файлов по папкам
bool ListDirectoryContents(const char *sDir, ht_item_t **tbl)
{
    WIN32_FIND_DATA fdFile;
    HANDLE hFind = NULL;

    char sPath[PATHMAXSIZE];

    //Specify a file mask. *.* = We want everything!
    sprintf_s(sPath, PATHMAXSIZE, "%s\\*.*", sDir);

    if ((hFind = FindFirstFile(sPath, &fdFile)) == INVALID_HANDLE_VALUE)
    {
        return false;
    }

    do
    {
        //Find first file will always return "."
        // and ".." as the first two directories.
        if (strcmp(fdFile.cFileName, ".") != 0
            && strcmp(fdFile.cFileName, "..") != 0)
        {
            //Build up our file path using the passed in
            // [sDir] and the file/foldername we just found:
            sprintf_s(sPath, PATHMAXSIZE, "%s\\%s", sDir,
fdFile.cFileName);

            //Is the entity a File or Folder?
            if (fdFile.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
            {
                // printf("Directory: %s\n", sPath);
                ListDirectoryContents(sPath, tbl); //Recursion, I love
it!
            }
            else {
                // printf("File: %s\n", sPath);
                ht_add(tbl, sPath, fgetcrc32(sPath));
            }
        }
    } while (FindNextFile(hFind, &fdFile)); //Find the next file.

    FindClose(hFind); //Always, Always, clean things up!

    return true;
}

```

```

////////////////////////////////////
//*****
//main.c - реализация GUI
//*****
////////////////////////////////////

#include <stdio.h>
#include "hashfunc.h"
#include "filehash.h"
#include <string.h>
#include <tchar.h>
#include <Windows.h>
#include "hashtable.h"
#include "dirview.h"
#include <Windowsx.h>
#include <CommCtrl.h>

#define WND_HEIGHT 300
#define WND_WIDTH 350
#define WND_CH_HEIGHT 150
#define WND_RS_HEIGHT 470
#define WND_RS_WIDTH 600
#define ScreenX GetSystemMetrics(SM_CXSCREEN)
#define ScreenY GetSystemMetrics(SM_CYSCREEN)

#define BUFF_LEN 26*4

#define MENU_EXIT 31
#define MENU_SCAN 27
#define MENU_IMPRINT 28
#define MENU_CHOOSDRIVE 29
#define MENU_ABOUT 30
#define IMPRWND_CANCEL 26
#define IMPRWND_START 25
#define MENU_RESULTS 24
#define LIBO_CHNGD 23

#define UNUCODE
#define _UNICODE

HWND liboNew, liboChanged, liboDeleted;
HMENU hMenu;
HWND cbDrivesImpr, cbDrivesScan;
HWND hPlain;
HINSTANCE My_hInst;
HWND MainWnd, ImprintWnd, ScanWnd, ResultsWnd;

LRESULT CALLBACK wndScanProcedure(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK WindowProcedure(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK wndImprintProcedure(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK wndResultsProcedure(HWND, UINT, WPARAM, LPARAM);
void GetImprint(HWND);
void GetScan(HWND);
void SetResultsControls(HWND);
void SetImprControls(HWND);
void SetScanControls(HWND);
void SetControls(HWND);
void SetImprintWnd(void);
void SetScanWnd(void);
void SetResultsWnd(void);
void InitMsg(HWND);

//Точка входа программы
int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst,
LPSTR args, int ncmdshow)

```

```

{
    My_hInst = hInst;
    WNDCLASSW wc = { 0 };
    wc.hbrBackground = (HBRUSH)COLOR_WINDOW;
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hInstance = hInst;
    wc.lpszClassName = L"myWindowClass";
    wc.lpfnWndProc = WindowProcedure;

    if (!RegisterClassW(&wc))
        return -1;

    InitMsg(MainWnd);

    MainWnd = CreateWindowW(wc.lpszClassName, L"Inspector",
        WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_MINIMIZEBOX | WS_VISIBLE,
        (ScreenX - WND_WIDTH) / 2,
        (ScreenY - WND_HEIGHT) / 2, WND_WIDTH, WND_HEIGHT, NULL, NULL,
        My_hInst, NULL);

    SetImprintWnd();
    SetScanWnd();
    SetResultsWnd();

    MSG msg = {0};

    while (GetMessageW(&msg, 0, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return 0;
}

//Сообщение при старте программы
void InitMsg(HWND hWnd)
{
    //СДЕЛАТЬ ДИНАМИЧЕСКИМ
    WCHAR Buff[BUFF_LEN];

    memset(Buff, 0, BUFF_LEN);

    GetLogicalDriveStringsW(BUFF_LEN, Buff);

    WCHAR *ptr;
    WCHAR drive_str[10] = L"";
    WCHAR str_to_outp[100] = L"Drives defined: ";
    ptr = Buff;

    SendMessageW(cbDrivesScan, (UINT)CB_ADDSTRING, (LPARAM)0, (LPARAM)L"All");
    while (*ptr) {
        wscat_s(drive_str, 10, ptr);
        wscat_s(str_to_outp, 100, ptr);
        wscat_s(str_to_outp, 100, L" , ");
        wcsncpy_s(drive_str, 10, L"");
        ptr += wcslen(ptr) + 1;
    }

    MessageBoxW(hWnd, str_to_outp, L"Drives", MB_OK);
    return;
}

```

```

//Создание элементов окна
void SetControls(HWND hWnd)
{
    //Buttons
    CreateWindowW(L"Button", L"Imprint", WS_CHILD | WS_VISIBLE |
BS_DEFPUSHBUTTON, 200, 30, 100,
    30, hWnd, (HMENU)MENU_IMPRINT, NULL, NULL);

    CreateWindowW(L"Button", L"Scan", WS_CHILD | WS_VISIBLE | BS_DEFPUSHBUTTON,
200, 70, 100,
    30, hWnd, (HMENU)MENU_SCAN, NULL, NULL);

    CreateWindowW(L"Static", L"", WS_CHILD | WS_VISIBLE | WS_BORDER, 0, 0, 150,
WND_HEIGHT, hWnd, NULL, NULL, NULL);

    CreateWindowW(L"button", L"Show Results", WS_CHILD | WS_VISIBLE |
BS_DEFPUSHBUTTON, 200, 170, 100,
    30, hWnd, (HMENU)MENU_RESULTS, NULL, NULL);

    //Drives:
    CreateWindowW(L"Static", L"Drives:", WS_CHILD | WS_VISIBLE, 25, 10, 60,
    20, hWnd, NULL, NULL, NULL);

    //СДЕЛАТЬ ДИНАМИЧЕСКИМ
    WCHAR Buff[BUFF_LEN];

    memset(Buff, 0, BUFF_LEN);

    GetLogicalDriveStringsW(BUFF_LEN, Buff);

    WCHAR *ptr;
    WCHAR drive_str[10] = L"- ";
    int y = 40;

    ptr = Buff;

    while (*ptr && y + 30 < WND_HEIGHT) {
        wcscat_s(drive_str, 10, ptr);
        CreateWindowW(L"Static", drive_str, WS_CHILD | WS_VISIBLE, 50, y, 60,
    20, hWnd, NULL, NULL, NULL);

        wcsncpy_s(drive_str, 10, L"- ");
        ptr += wcslen(ptr) + 1;
        y += 30;
    }
}

//Главное меню программы
void MainMenu(HWND hWnd)
{
    hMenu = CreateMenu();
    HMENU hFileMenu = CreateMenu();
    HMENU hAboutMenu = CreateMenu();
    //Menu File:
    AppendMenuW(hMenu, MF_POPUP, (UINT_PTR)hFileMenu, L"File");
    AppendMenuW(hFileMenu, MF_POPUP, MENU_IMPRINT, L"Imprint...");
    AppendMenuW(hFileMenu, MF_STRING, MENU_SCAN, L"Scan...");
    AppendMenuW(hFileMenu, MF_SEPARATOR, (UINT_PTR)NULL, NULL);
    AppendMenuW(hFileMenu, MF_STRING, MENU_EXIT, L"Exit");

    // Menu About:
    AppendMenuW(hMenu, MF_POPUP, (UINT_PTR)hAboutMenu, L"About");
}

```

```

        AppendMenuW(hAboutMenu, MF_STRING, MENU_ABOUT, L"Inspector");

        SetMenu(hWnd, hMenu);
    }

    //Создание элементов на окне Results
    void SetResultsControls(HWND hWnd)
    {
        //Labels

        CreateWindowW(L"Static", L"New:", WS_CHILD | WS_VISIBLE | WS_BORDER, 5, 5,
WND_RS_WIDTH - 26,
        137, hWnd, NULL, NULL, NULL);

        CreateWindowW(L"Static", L"Changed:", WS_CHILD | WS_VISIBLE | WS_BORDER, 5,
145, WND_RS_WIDTH - 26,
        137, hWnd, NULL, NULL, NULL);

        CreateWindowW(L"Static", L"Deleted:", WS_CHILD | WS_VISIBLE | WS_BORDER, 5,
285, WND_RS_WIDTH - 26,
        137, hWnd, NULL, NULL, NULL);

        // Adding a ListBoxes.
        liboNew = CreateWindowExW(WS_EX_CLIENTEDGE,
            L"LISTBOX",
            NULL,
            WS_CHILD | WS_VISIBLE | ES_AUTOVSCROLL | WS_VSCROLL | WS_HSCROLL,
            7,
            25,
            WND_RS_WIDTH - 30,
            120,
            hWnd,
            NULL,
            My_hInst,
            NULL);

        liboChanged = CreateWindowExW(WS_EX_CLIENTEDGE,
            L"LISTBOX",
            NULL,
            WS_CHILD | WS_VISIBLE | ES_AUTOVSCROLL | WS_VSCROLL | WS_HSCROLL,
            7,
            165,
            WND_RS_WIDTH - 30,
            120,
            hWnd,
            LIBO_CHNGD,
            My_hInst,
            NULL);

        liboDeleted = CreateWindowExW(WS_EX_CLIENTEDGE,
            L"LISTBOX",
            NULL,
            WS_CHILD | WS_VISIBLE | ES_AUTOVSCROLL | WS_VSCROLL | WS_HSCROLL,
            7,
            305,
            WND_RS_WIDTH - 30,
            120,
            hWnd,
            NULL,
            My_hInst,
            NULL);

        SendMessageW(liboDeleted, LB_SETHORIZONTALEXTENT, (LPARAM)PATHMAXSIZE,
(LPARAM)0);
    }

```

```

        SendMessageW(liboNew, LB_SETHORIZONTALEXTENT, (WPARAM)PATHMAXSIZE,
(LPARAM)0);
        SendMessageW(liboChanged, LB_SETHORIZONTALEXTENT, (WPARAM)PATHMAXSIZE,
(LPARAM)0);
    }

    //Создание элементов окна Imprint
    void SetImprControls(HWND hWnd)
    {

        CreateWindowW(L"Static", L"Drives:", WS_CHILD | WS_VISIBLE, 25, 10, 60,
            20, hWnd, NULL, NULL, NULL);

        CreateWindowW(L"Button", L"Start", WS_CHILD | WS_VISIBLE, 135, 45, 80,
            25, hWnd, (HMENU)IMPRWND_START, NULL, NULL);

        CreateWindowW(L"Button", L"Cancel", WS_CHILD | WS_VISIBLE, 235, 45, 80,
            25, hWnd, (HMENU)IMPRWND_CANCEL, NULL, NULL);

        cbDrivesImpr = CreateWindowW(L"Combobox", L"",
            CBS_DROPDOWNLIST | CBS_HASSTRINGS | WS_CHILD | WS_OVERLAPPED |
WS_VISIBLE, 25, 46, 100,
            300, hWnd, NULL, NULL, NULL);

        //СДЕЛАТЬ ДИНАМИЧЕСКИМ
        WCHAR Buff[BUFF_LEN];

        memset(Buff, 0, BUFF_LEN);

        GetLogicalDriveStringsW(BUFF_LEN, Buff);

        WCHAR *ptr;
        WCHAR drive_str[10] = L"";

        ptr = Buff;

        SendMessageW(cbDrivesImpr, (UINT)CB_ADDSTRING, (WPARAM)0, (LPARAM)L"All");
        while (*ptr) {
            wscat_s(drive_str, 10, ptr);

            SendMessageW(cbDrivesImpr, (UINT)CB_ADDSTRING, (WPARAM)0,
(LPARAM)drive_str);
            wcscpy_s(drive_str, 10, L "");
            ptr += wcslen(ptr) + 1;
        }

        //free(Buff);
        //All - default choice
        SendMessageW(cbDrivesImpr, (UINT)CB_SETCURSEL, (WPARAM)0, (LPARAM)0);
    }

    //Создание элементов окна Scan
    void SetScanControls(HWND hWnd)
    {

        CreateWindowW(L"Static", L"Drives:", WS_CHILD | WS_VISIBLE, 25, 10, 60,
            20, hWnd, NULL, NULL, NULL);

        CreateWindowW(L"Button", L"Start", WS_CHILD | WS_VISIBLE, 135, 45, 80,
            25, hWnd, (HMENU)IMPRWND_START, NULL, NULL);

        CreateWindowW(L"Button", L"Cancel", WS_CHILD | WS_VISIBLE, 235, 45, 80,
            25, hWnd, (HMENU)IMPRWND_CANCEL, NULL, NULL);
    }

```

```

        cbDrivesScan = CreateWindowW(L"Combobox", L"",
            CBS_DROPDOWNLIST | CBS_HASSTRINGS | WS_CHILD | WS_OVERLAPPED |
WS_VISIBLE, 25, 46, 100,
            300, hWnd, NULL, NULL, NULL);

        //СДЕЛАТЬ ДИНАМИЧЕСКИМ
        WCHAR Buff[BUFF_LEN];

        memset(Buff, 0, BUFF_LEN);

        GetLogicalDriveStringsW(BUFF_LEN, Buff);

        WCHAR *ptr;
        WCHAR drive_str[10] = L"";

        ptr = Buff;

        SendMessageW(cbDrivesScan, (UINT)CB_ADDSTRING, (WPARAM)0, (LPARAM)L"All");
        while (*ptr) {
            wscat_s(drive_str, 10, ptr);

            SendMessageW(cbDrivesScan, (UINT)CB_ADDSTRING, (WPARAM)0,
(LPARAM)drive_str);
            wcscpy_s(drive_str, 10, L "");
            ptr += wcslen(ptr) + 1;
        }

        //free(Buff);
        //All - default choice
        SendMessageW(cbDrivesScan, (UINT)CB_SETCURSEL, (WPARAM)0, (LPARAM)0);
    }

    //Обработка сообщений окна Scan
    LRESULT CALLBACK wndScanProcedure(HWND hWnd, UINT msg,
        WPARAM wp, LPARAM lp)
    {
        switch (msg) {
            case WM_COMMAND:
                switch (wp) {
                    case IMPRWND_CANCEL:
                        ShowWindow(hWnd, SW_HIDE);
                        ShowWindow(MainWnd, SW_RESTORE);
                        break;
                    case IMPRWND_START:
                        GetScan(cbDrivesScan);
                        break;
                }
                break;
            case WM_CREATE:
                SetScanControls(hWnd);
                break;
            case WM_CLOSE:
                ShowWindow(hWnd, SW_HIDE);
                ShowWindow(MainWnd, SW_RESTORE);
                break;
            case WM_DESTROY:
                break;
            default:
                return DefWindowProcW(hWnd, msg, wp, lp);
        }
        return 0;
    }

    //Обработка сообщений окна Imprint

```

```

LRESULT CALLBACK wndImprintProcedure(HWND hWnd, UINT msg,
    WPARAM wp, LPARAM lp)
{
    switch (msg) {
    case WM_COMMAND:
        switch (wp) {
            case IMPRWND_CANCEL:
                ShowWindow(hWnd, SW_HIDE);
                ShowWindow(MainWnd, SW_RESTORE);
                break;
            case IMPRWND_START:
                GetImprint(cbDrivesImpr);
                break;
        }
        break;
    case WM_CREATE:
        SetImprControls(hWnd);
        break;
    case WM_CLOSE:
        ShowWindow(hWnd, SW_HIDE);
        ShowWindow(MainWnd, SW_RESTORE);
        break;
    case WM_DESTROY:
        break;
    default:
        return DefWindowProcW(hWnd, msg, wp, lp);
    }
    return 0;
}

//Обработка сообщений окна Results
LRESULT CALLBACK wndResultsProcedure(HWND hWnd, UINT msg,
    WPARAM wp, LPARAM lp)
{
    switch (msg) {
    case WM_COMMAND:
        switch (wp) {
            case LBN_SETFOCUS:
                SendMessageW(liboChanged, (UINT)CB_SETCURSEL, (WPARAM)-1,
(LPARAM)0);
                break;
        }
        break;
    case WM_CREATE:
        SetResultsControls(hWnd);
        break;
    case WM_CLOSE:
        ShowWindow(hWnd, SW_HIDE);
        ShowWindow(MainWnd, SW_RESTORE);
        break;
    case WM_DESTROY:
        break;
    default:
        return DefWindowProcW(hWnd, msg, wp, lp);
    }
    return 0;
}

//Обработка сообщений главного окна
LRESULT CALLBACK WindowProcedure(HWND hWnd, UINT msg,
    WPARAM wp, LPARAM lp)
{
    switch (msg) {
    case WM_COMMAND:
        switch (wp) {

```



```

        case MENU_IMPRINT:
            ShowWindow(hWnd, SW_HIDE);
            ShowWindow(ImprintWnd, SW_RESTORE);
            break;
        case MENU_SCAN:
            ShowWindow(hWnd, SW_HIDE);
            ShowWindow(ScanWnd, SW_RESTORE);
            break;
        case MENU_RESULTS:
            ShowWindow(hWnd, SW_HIDE);
            ShowWindow(ResultsWnd, SW_RESTORE);
            break;
        case MENU_EXIT:
            MessageBeep(MB_OK);
            if (MessageBoxW(hWnd, L"Are you sure?", L"Exit", MB_YESNO) ==
IDYES)
                DestroyWindow(hWnd);
            break;
        case MENU_ABOUT:
            MessageBoxW(NULL, L"Program inspector", L"Info", MB_OK);
            break;
    }
    break;
case WM_CREATE:
    MainMenu(hWnd);
    SetControls(hWnd);
    break;
case WM_CLOSE:
    MessageBeep(MB_OK);
    if (MessageBoxW(hWnd, L"Are you sure?", L"Exit", MB_YESNO) == IDYES)
        DestroyWindow(hWnd);
    break;
case WM_DESTROY:
    PostQuitMessage(0);
    break;
default:
    return DefWindowProcW(hWnd, msg, wp, lp);
}
return 0;
}

//Создание окна Results
void SetResultsWnd(void) {
    WNDCLASSW wc;
    memset(&wc, 0, sizeof(WNDCLASSW));
    wc.hbrBackground = (HBRUSH)COLOR_WINDOW;
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hInstance = My_hInst;
    wc.lpszClassName = L"myResultsClass";
    wc.lpfnWndProc = wndResultsProcedure;

    if (!RegisterClassW(&wc))
        return;

    ResultsWnd = CreateWindowW(wc.lpszClassName, L"Results",
        WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_MINIMIZEBOX, (ScreenX -
WND_RS_WIDTH) / 2,
        (ScreenY - WND_RS_HEIGHT) / 2, WND_RS_WIDTH, WND_RS_HEIGHT, NULL,
NULL, My_hInst, NULL);
}

//Создание окна Imprint
void SetImprintWnd(void) {
    WNDCLASSW wc;
    memset(&wc, 0, sizeof(WNDCLASSW));

```

```

        wc.hbrBackground = (HBRUSH)COLOR_WINDOW;
        wc.hCursor = LoadCursor(NULL, IDC_ARROW);
        wc.hInstance = My_hInst;
        wc.lpszClassName = L"myChildClass";
        wc.lpfnWndProc = wndImprintProcedure;

        if (!RegisterClassW(&wc))
            return;

        ImprintWnd = CreateWindowW(wc.lpszClassName, L"Imprint",
            WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_MINIMIZEBOX, (ScreenX -
WND_WIDTH) / 2,
            (ScreenY - WND_CH_HEIGHT) / 2, WND_WIDTH, WND_CH_HEIGHT, NULL, NULL,
My_hInst, NULL);
    }

    //Создание окна Scan
    void SetScanWnd(void) {
        WNDCLASSW wc;
        memset(&wc, 0, sizeof(WNDCLASSW));
        wc.hbrBackground = (HBRUSH)COLOR_WINDOW;
        wc.hCursor = LoadCursor(NULL, IDC_ARROW);
        wc.hInstance = My_hInst;
        wc.lpszClassName = L"myScanClass";
        wc.lpfnWndProc = wndScanProcedure;

        if (!RegisterClassW(&wc))
            return;

        ScanWnd = CreateWindowW(wc.lpszClassName, L"Scan",
            WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_MINIMIZEBOX, (ScreenX -
WND_WIDTH) / 2,
            (ScreenY - WND_CH_HEIGHT) / 2, WND_WIDTH, WND_CH_HEIGHT, NULL, NULL,
My_hInst, NULL);
    }

    //Создание слепка диска
    void GetImprint(HWND cb)
    {
        WCHAR drive_str[10] = L"";
        char ch_drive_str[10] = "";
        char out_fname[PATHMAXSIZE];
        uint32_t res = 0;
        WCHAR str_res[40];

        htable_t *h_table = _ht_init();

        memset(drive_str, '\\0', 10*sizeof(WCHAR));

        LRESULT i = SendMessageW(cb, (UINT)CB_GETCURSEL, (WPARAM)0, (LPARAM)0);

        SendMessageW(cb, (UINT)CB_GETLBTEXT, (WPARAM)i, (LPARAM)drive_str);

        size_t len = wcstombs(ch_drive_str, drive_str, wcslen(drive_str));
        if (len > 0u)
            ch_drive_str[len] = '\\0';

        if (wcscmp(drive_str, L"All") == 0) {
            MessageBeep(MB_OK);
            MessageBoxW(cb, L"I can't find any files...", L"Imprint", MB_OK);
        }
        else {
            ListDirectoryContents(ch_drive_str, h_table->ht_items);
            strcat_s(ch_drive_str, 10, ".csm");
            fname_to_str(ch_drive_str);
        }
    }

```

```

        res = (uint32_t)fhash_save(h_table->ht_items, ch_drive_str);
    }
    ht_destroy(h_table->ht_items);
    free(h_table);
    swprintf_s(str_res, 40, L"%u files was imprinted!", res);
    MessageBoxW(NULL, str_res, L"Imprint", MB_OK);
    MessageBeep(MB_OK);
    SendMessageW(ImprintWnd, WM_CLOSE, (WPARAM)0, (LPARAM)0);
}

// Сканирование диска
void GetScan(HWND cb)
{
    // Указатели на измененные файлы
    ht_item_t *tmp_changed = NULL, *tmp_deleted = NULL, *tmp_new = NULL,
    *ptr_to_del = NULL;

    WCHAR drive_str[10] = L"";
    char ch_drive_str[10] = "", _ch_drive_srt[10] = "";
    char out_fname[PATHMAXSIZE];
    uint32_t res = 0;
    WCHAR str_res[40];

    memset(drive_str, '\\0', 10 * sizeof(WCHAR));

    LRESULT i = SendMessageW(cb, (UINT)CB_GETCURSEL, (WPARAM)0, (LPARAM)0);

    SendMessageW(cb, (UINT)CB_GETLBTEXT, (WPARAM)i, (LPARAM)drive_str);

    size_t len = wcstombs(ch_drive_str, drive_str, wcslen(drive_str));
    if (len > 0u)
        ch_drive_str[len] = '\\0';

    if (wcscmp(drive_str, L"All") == 0) {
        MessageBeep(MB_OK);
        MessageBoxW(NULL, L"I can't find any files...", L"Imprint", MB_OK);
    }
    else {
        strcpy_s(_ch_drive_srt, 10, ch_drive_str);
        fname_to_str(ch_drive_str);
        strcat_s(ch_drive_str, 10, ".csm");
        if (_ht_chngs(&tmp_new, &tmp_deleted, &tmp_changed, ch_drive_str,
        _ch_drive_srt) == -1) {
            MessageBeep(MB_OK);
            if (MessageBoxW(NULL, L"There is not imprint of this
            file.\\nMake it?", L"No Imprint", MB_YESNO) == IDYES) {
                GetImprint(cbDrivesScan);
                SendMessageW(ScanWnd, WM_CLOSE, (WPARAM)0, (LPARAM)0);
            }
        }
        else {
            SendMessageW(liboChanged, LB_RESETCONTENT, (WPARAM)0,
            (LPARAM)0);

            SendMessageW(liboNew, LB_RESETCONTENT, (WPARAM)0, (LPARAM)0);
            SendMessageW(liboDeleted, LB_RESETCONTENT, (WPARAM)0,
            (LPARAM)0);

            while (tmp_deleted) {
                ptr_to_del = tmp_deleted;
                SendMessageA(liboDeleted, LB_ADDSTRING, (WPARAM)0,
                (LPARAM)tmp_deleted->fname);

                tmp_deleted = tmp_deleted->pnext;
                free(ptr_to_del);
            }
            free(tmp_deleted);
            while (tmp_changed) {

```

```

ptr_to_del = tmp_changed;
SendMessageA(liboChanged, LB_ADDSTRING, (WPARAM)0,
(LPARAM)tmp_changed->fname);
tmp_changed = tmp_changed->pnext;
free(ptr_to_del);
}
free(tmp_changed);
while (tmp_new) {
ptr_to_del = tmp_new;
SendMessageA(liboNew, LB_ADDSTRING, (WPARAM)0,
(LPARAM)tmp_new->fname);
tmp_new = tmp_new->pnext;
free(ptr_to_del);
}
free(tmp_new);
MessageBeep(MB_OK);
if (MessageBoxW(NULL, L"Scan ends.\nDo you want to see results
now?", L"Scan", MB_YESNO) == IDYES) {
SendMessageW(ScanWnd, WM_CLOSE, (WPARAM)0, (LPARAM)0);
SendMessageW(MainWnd, WM_COMMAND, (WPARAM)MENU_RESULTS,
(LPARAM)0);
}
}
}
}

```

Обозначение				Наименование				Дополнительные сведения			
				<u>Текстовые документы</u>							
БГУИР КП 1–40 01 01 18 ПЗ				Пояснительная записка				46 с.			
				<u>Графические документы</u>							
ГУИР 751005 18 ПД				"Программа-ревизор" Схема работы системы.				Формат А1			