

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра информационной безопасности**

**КУРСОВАЯ РАБОТА**  
**по дисциплине «Программирование»**  
**Тема: Приложение «Сигнатурный сканер»**

Студент гр.

---

Преподаватель

---

Санкт-Петербург

2021

## ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент

Группа

Тема работы: Приложение «Сигнатурный сканер».

Исходные данные: разработать на языке программирования C или C++ (по выбору студента) сигнатурный сканер. Сигнатурный сканер должен сканировать файлы и обнаруживать заданные образцы по наличию сигнатуры в файле. Минимальный размер сигнатуры должна составлять 8 байт, поиск сигнатуры необходимо осуществлять по фиксированному смещению. При поиске сигнатуры в файле необходимо учитывать формат файла. В сигнатурном сканере необходимо реализовать поддержку формата PE (Portable Executable). Если сигнатура обнаружена в файле, то необходимо вывести имя файла с указанием того, какому образцу соответствует найденная сигнатура. Информация о сигнатурах должна храниться в отдельном файле и считываться сканером при запуске. Путь к файлу, в котором хранится сигнатура, вводится пользователем. Путь к файлу для проверки вводится пользователем. Приложение должно иметь консольный или графический интерфейс, по выбору студента. Интерфейс приложения должен быть интуитивно понятным и содержать подсказки для пользователя. В исходном коде приложения должны быть реализованы проверки аргументов реализованных студентом функций и проверки возвращаемых функциями значений (для всех функций — как сторонних, так и реализованных студентом). Приложение должно корректно обрабатывать ошибки, в том числе ошибки ввода/вывода, выделения/освобождения памяти и т. д.

Содержание пояснительной записки: введение, теоретическая часть, классификация компьютерных вирусов, основные каналы распространения вирусов, обнаружение вирусов, основанное на сигнатурах, виды антивирусных программ, реализация программы, использованное ПО, описание функций, результаты работы программы, заключение, список использованных источников, приложение 1 – руководство пользователя, приложение 2 – блок-схема алгоритма, приложение 3 – исходный код.

Предполагаемый объем пояснительной записки:

Не менее 40 страниц.

Дата выдачи задания: 25.02.2021

Дата сдачи реферата: 07.06.2021

Дата защиты реферата: 11.06.2021

Студент

---

Преподаватель

---

## **АННОТАЦИЯ**

Приложение «Сигнатурный сканер» является несложной программой. Его реализация в качестве курсовой работы помогает освоить основные теоретические знания по дисциплине «программирование», развить навыки написания кода на языке программирования C/C++, закрепить знания по работе с файлами.

Программа способна проверять любой файл на вредоносность, выводить в консоли сообщения о результатах проверки.

## **SUMMARY**

The «Signature Scanner» application is a simple program. Its implementation as a course work helps to master the basic theoretical knowledge of the discipline "programming", to develop the skills of writing code in the C/C++ programming language, to consolidate the knowledge of working with files.

The program is able to check any file for malware, display messages in the console about the results of the check.

## СОДЕРЖАНИЕ

	Введение	6
1.	Теоретическая часть	7
1.1.	Классификация компьютерных вирусов	7
1.2.	Основные каналы распространения вирусов	9
1.3.	Обнаружение вирусов, основанное на сигнатурах	12
1.4.	Виды антивирусных программ	13
2.	Реализация программы	17
2.1.	Использованное ПО	17
2.2.	Описание функций	17
3.	Результаты работы программы	24
	Заключение	27
	Список использованных источников	28
	Приложение 1. Руководство пользователя	29
	Приложение 2. Блок-схема алгоритма	34
	Приложение 3. Исходный код	35

## **ВВЕДЕНИЕ**

Тема данной курсовой работы очень актуальна. Это обусловлено тем, что сигнатурный сканер является фундаментом работы любой антивирусной программы, и с развитием Интернета проблема внешних угроз стала, как никогда, острой, новые вирусы и другие вредоносные программы появляются практически каждый день.

В курсовой работе нам была поставлена задача научиться проверять файлы на содержание той или иной сигнатуры, сообщать пользователю о результатах.

Разрешить данную задачу получилось посредством написания консольного приложения, для работы которого необходимо иметь антивирусную базу с вредоносной сигнатурой.

В курсовой работе удалось достигнуть успешного написания рабочего кода, который способен проверять любой файл на вредоносность, выводить в консоли сообщения о состоянии проверки.

Подробнее о работе антивирусных программ можно прочитать в последующем изложении теоретического материала.

# **1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

## **1.1. Классификация компьютерных вирусов**

На сегодняшний день известны десятки тысяч различных компьютерных вирусов. Невзирая на такое изобилие, количество видов вирусов, различающихся механизмом распространения и принципом действия, довольно ограничено. Существуют также комбинированные вирусы, которые можно отнести одновременно к нескольким типам. Вирусы можно разбить на классы: по среде обитания, операционной системе (ОС), особенностям алгоритма работы, деструктивным возможностям.

Основной и наиболее распространенной систематизацией компьютерных вирусов является классификация согласно среде обитания, или по типам объектов компьютерной системы, в которые внедряются вирусы. По среде обитания компьютерные вирусы можно разделить на: файловые, загрузочные, макровирусы, сетевые.

Файловые вирусы либо внедряются в выполняемые файлы (наиболее распространенный тип вирусов) различными способами, либо создают файлы-двойники (компаньон-вирусы), либо используют характерные черты организации файловой системы (link-вирусы).

Загрузочные вирусы записывают себя либо в загрузочный сектор диска (boot-сектор), либо в сектор, содержащий системный загрузчик винчестера (Master Boot Record). Загрузочные вирусы замещают код программы, получающей управление при загрузке системы. В результате при перезагрузке управление передается вирусу. При этом оригинальный boot-сектор обычно переносится в какой-либо другой сектор диска. Иногда загрузочные вирусы называют бутовыми вирусами.

Макровирусы заражают макропрограммы и файлы документов современных систем обработки информации, в частности файлы-документы и электронные таблицы популярных редакторов Microsoft Word, Microsoft Excel и др. Для размножения макровирусы используют возможности

макроязыков и при их помощи переносят себя из одного зараженного файла в другие. Вирусы этого типа получают управление при открытии зараженного файла и инфицируют файлы, к которым впоследствии идет обращение из соответствующего офисного приложения.

Сетевые вирусы используют для своего распространения протоколы или команды компьютерных сетей и электронной почты. Иногда сетевые вирусы называют программами типа «червь». Сетевые черви подразделяются на Internet-черви (распространяются по Internet), LAN-черви (распространяются по локальной сети), IRC-черви Internet Relay Chat (распространяются через чаты). Существуют также смешанные типы, которые совмещают в себе сразу несколько технологий.

Существуют много комбинированных типов компьютерных вирусов, например, известен сетевой макро-вирус, который заражает редактируемые документы, а также рассылает свои копии по электронной почте. В качестве другого примера вирусов комбинированного типа можно указать файлово-загрузочные вирусы, заражающие как файлы, так и загрузочные секторы дисков. Такие вирусы имеют усложненный алгоритм работы и применяют своеобразные методы проникновения в систему.

Другим критерием деления компьютерных вирусов на классы является операционная система, объекты которой подвергаются заражению. Каждый файловый или сетевой вирус заражает файлы какой-либо одной или нескольких ОС — DOS, Windows 95/98, Windows NT/2000 и т. д. Макро-вирусы заражают файлы форматов Word, Excel, Microsoft Office. На определенные форматы расположения системных данных в загрузочных секторах дисков также ориентированы загрузочные вирусы.

Конечно, эти методы классификации не являются единственными, существуют много различных схем типизации вирусов. Однако ограничимся пока классификацией компьютерных вирусов по среде обитания, поскольку она является базовой, и перейдем к рассмотрению общих принципов функционирования вирусов. Анализ основных этапов «жизненного цикла»



этих вредоносных программ позволяет выделить их различные признаки и особенности, которые могут быть положены в основу дополнительных классификаций.

## **1.2. Основные каналы распространения вирусов**

Для того чтобы создать эффективную систему антивирусной защиты компьютеров и корпоративных сетей, необходимо четко представлять себе, откуда грозит опасность. Вирусы находят самые разные каналы распространения, причем к старым способам постоянно добавляются новые. Рассмотрим классические способы.

Файловые вирусы распространяются вместе с файлами программ в результате обмена дискетами и программами, загрузки программ из сетевых каталогов, с Web- или ftp-серверов. Загрузочные вирусы попадают на компьютер, когда пользователь забывает зараженную дискету в дисковом, а затем перезагружает ОС. Загрузочный вирус также может быть занесен на компьютер вирусами других типов. Макрокомандные вирусы распространяются в результате обмена зараженными файлами офисных документов, такими как файлы Microsoft Word, Excel, Access. Если зараженный компьютер подключен к локальной сети, вирус легко может оказаться на дисках файл-сервера, а оттуда через каталоги, доступные для записи, попасть на все остальные компьютеры сети. Так начинается вирусная эпидемия. Системному администратору следует помнить, что вирус имеет в сети такие же права, что и пользователь, на компьютер которого этот вирус пробрался. Поэтому он может попасть во все сетевые каталоги, доступные пользователю. Если же вирус завелся на рабочей станции администратора сети, последствия могут быть очень тяжелыми.

В настоящее время глобальная сеть Internet является основным источником вирусов. Большое число заражений вирусами происходит при обмене письмами по электронной почте в форматах Microsoft Word. Электронная почта служит каналом распространения макрокомандных

вирусов, так как вместе с сообщениями часто отправляются офисные документы.

Заражения вирусами могут осуществляться как непреднамеренно, так и по злому умыслу. Например, пользователь зараженного макровирусом редактора, сам того не подозревая, может рассылать зараженные письма адресатам, которые в свою очередь отправляют новые зараженные письма и т. д. С другой стороны, злоумышленник может преднамеренно послать по электронной почте вместе с вложенным файлом исполняемый модуль вирусной или троянской программы, вредоносный программный сценарий Visual Basic Script, зараженную или троянскую программу сохранения экрана монитора, словом — любой опасный программный код.

Распространители вирусов часто пользуются для маскировки тем фактом, что диалоговая оболочка Microsoft Windows по умолчанию не отображает расширения зарегистрированных файлов. Например, файл с именем FreeCreditCard.txt.exe, будет показан пользователю как FreeCreditCard.txt. Если пользователь попытается открыть такой файл, будет запущена вредоносная программа.

Сообщения электронной почты часто приходят в виде документов HTML, которые могут включать ссылки на элементы управления ActiveX, апплеты Java и другие активные компоненты. Из-за ошибок в почтовых клиентах злоумышленники могут воспользоваться такими активными компонентами для внедрения вирусов и троянских программ на компьютеры пользователей. При получении сообщения в формате HTML почтовый клиент показывает его содержимое в своем окне. Если сообщение содержит вредоносные активные компоненты, они сразу же запускаются и выполняют заложенные в них функции. Чаще всего таким способом распространяются троянские программы и черви.

Пользователи могут «получить» вирус или троянскую программу во время простого серфинга сайтов Интернета, посетив троянский Web-сайт. Ошибки в браузерах пользователей зачастую приводят к тому, что активные

компоненты троянских Web-сайтов (элементы управления ActiveX или апплеты Java) внедряют на компьютеры пользователей вредоносные программы. Здесь используется тот же самый механизм, что и при получении сообщений электронной почты в формате HTML. Но заражение происходит незаметно: активные компоненты Web-страниц могут внешне никак себя не проявлять. Приглашение посетить троянский сайт пользователь может получить в обычном электронном письме.

Локальные сети также представляют собой путь быстрого заражения. Если не принимать необходимых мер защиты, то зараженная рабочая станция при входе в локальную сеть заражает один или несколько служебных файлов на сервере. В качестве таких файлов могут выступать служебный файл LOGIN.COM, Excel-таблицы и стандартные документы-шаблоны, применяемые в фирме. Пользователи при входе в эту сеть запускают зараженные файлы с сервера, и в результате вирус получает доступ на компьютеры пользователей. Другие каналы распространения вредоносных программ

Одним из серьезных каналов распространения вирусов являются пиратские копии ПО. Часто нелегальные копии на дискетах и CD-дисках содержат файлы, зараженные разнообразными типами вирусов. К источникам распространения вирусов следует также отнести электронные конференции и файл-серверы ftp и BBS. Часто авторы вирусов закладывают зараженные файлы сразу на несколько файл-серверов ftp/BBS или рассылают одновременно по нескольким электронным конференциям, причем зараженные файлы обычно маскируют под новые версии программных продуктов и даже антивирусов. Компьютеры, установленные в учебных заведениях и Интернет-центрах и работающие в режиме общего пользования, также могут легко оказаться источниками распространения вирусов. Если один из таких компьютеров оказался зараженным вирусом с дискеты очередного пользователя, тогда дискеты и всех остальных пользователей, работающих на этом компьютере, окажутся зараженными.

По мере развития компьютерных технологий совершенствуются и компьютерные вирусы, приспосабливаясь к новым для себя сферам обитания. В любой момент может появиться компьютерный вирус, троянская программа или «червь» нового, неизвестного ранее типа, либо известного типа, но нацеленного на новое компьютерное оборудование. Новые вирусы могут использовать неизвестные или не существовавшие ранее каналы распространения, а также новые технологии внедрения в компьютерные системы. Чтобы исключить угрозу вирусного заражения, системный администратор корпоративной сети должен внедрять методики антивирусной защиты и постоянно отслеживать новости в мире компьютерных вирусов.

### **1.3. Обнаружение вирусов, основанное на сигнатурах**

Метод обнаружения, основанный на сигнатурах, — это метод, при котором антивирусный сканер, просматривая файл, обращается к словарю с известными атаками, который составили и дополняют разработчики антивирусов. В случае соответствия какого-либо участка кода просматриваемой программы известному коду (сигнатуре) вируса в словаре антивирус удаляет инфицированный файл, пытается восстановить файл, удалив сам вирус из тела файла, или выполнить другие действия. Также антивирусная программа может отправить его в карантин, то есть делает невозможным его запуск во избежание дальнейшего распространения вируса.

Этот метод можно сравнить с паспортным контролем на границе, к примеру в аэропорту. Когда вы показываете свой паспорт, ваши паспортные данные проверяют в специальной базе разыскиваемых преступников, если их там нет, то вас благополучно пропускают.

Подобный способ обнаружения вирусов является старейшим. Первые антивирусные программы умели обнаруживать вирусы только таким способом, сверяя содержимое каждого файла со своим словарем. Современные антивирусные программы также используют сигнатурный анализ, однако он не является единственным средством обнаружения вирусов. На сегодняшний день этот метод малоэффективен.

Основным недостатком сигнатурного анализа является то, что он позволяет обнаруживать только уже известные вирусы. А вирусы, сигнатуры которых еще не занесены в словари, обнаружить таким способом не получится. Иногда, для того чтобы вирус смог проникнуть в сеть какой-то конкретной организации, его код разрабатывают специально таким образом, чтобы данной сигнатуры не было в словаре антивируса, используемого в этой организации. Так как при сигнатурном анализе антивирус ничего не знает об этом новом вирусе, вирус с успехом проникает в корпоративную сеть. Кроме того, разработчики вирусов специально шифруют и видоизменяют код вирусов, для того чтобы сигнатура этого кода отсутствовала в базе.

Еще одним существенным недостатком метода обнаружения, основанного на сигнатурах, является необходимость регулярного обновления сигнатур. Для такого обновления необходим доступ в Интернет. В случае если вы не обновляли свою базу антивирусных сигнатур хотя бы один месяц, вы подвергаете свой компьютер серьезной угрозе, так как за это время появились тысячи вирусов, неизвестных вашей антивирусной системе. Не забывайте об этом и настройте ежедневное обновление своего антивируса.

#### **1.4. Виды антивирусных программ**

Различают следующие виды антивирусных программ: программы-фаги (сканеры), программы-ревизоры (CRC-сканеры), программы-блокировщики, программы-иммунизаторы.

Программы-фаги (сканеры) используют для обнаружения вирусов метод сопоставления с эталоном, метод эвристического анализа и некоторые другие методы. Программы-фаги осуществляют поиск характерной для конкретного вируса маски путем сканирования в оперативной памяти и в файлах, и при обнаружении выдают соответствующее сообщение. Программы-фаги не только находят зараженные вирусами файлы, но и «лечат» их, т. е. удаляют из файла тело программы-вируса, возвращая файлы в исходное состояние. В начале работы программы-фаги сканируют оперативную память, обнаруживают вирусы и уничтожают их и только затем

переходят к «лечению» файлов. Среди фагов выделяют полифаги — программы-фаги, предназначенные для поиска и уничтожения большого числа вирусов.

Программы-фаги можно разделить на две категории — универсальные и специализированные сканеры. Универсальные сканеры рассчитаны на поиск и обезвреживание всех типов вирусов вне зависимости от ОС, на работу в которой рассчитан сканер. Специализированные сканеры предназначены для обезвреживания ограниченного числа вирусов или только одного их класса, например макровирусов. Специализированные сканеры, рассчитанные только на макровирусы, оказываются более удобным и надежным решением для защиты систем документооборота в средах MS Word и MS Excel.

Программы-фаги делятся также на резидентные мониторы, производящие сканирование «на лету», и нерезидентные сканеры, обеспечивающие проверку системы только по запросу. Резидентные мониторы обеспечивают более надежную защиту системы, поскольку они немедленно реагируют на появление вируса, в то время как нерезидентный сканер способен опознать вирус только во время своего очередного запуска.

К достоинствам программ-фагов всех типов относится их универсальность. К недостаткам следует отнести относительно небольшую скорость поиска вирусов и относительно большие размеры антивирусных баз.

Наиболее известные программы-фаги: Aidstest, Scan, Norton AntiVirus, Doctor Web. Учитывая, что постоянно появляются новые вирусы, программы-фаги быстро устаревают, и требуется регулярное обновление версий.

Программы-ревизоры (CRC-сканеры) используют для поиска вирусов метод обнаружения изменений. Принцип работы CRC-сканеров основан на подсчете CRC-сумм (кодов циклического контроля) для присутствующих на диске файлов/системных секторов. Эти CRC-суммы, а также некоторая

другая информация (длины файлов, даты их последней модификации и др.) затем сохраняются в БД антивируса. При последующем запуске CRC-сканеры сверяют данные, содержащиеся в БД, с реально подсчитанными значениями. Если информация о файле, записанная в БД, не совпадает с реальными значениями, то CRC-сканеры сигнализируют о том, что файл был изменен или заражен вирусом. Как правило, сравнение состояний производят сразу после загрузки ОС.

CRC-сканеры, использующие алгоритмы анти-стелс, являются довольно мощным средством против вирусов: практически 100 % вирусов оказываются обнаруженными почти сразу после их появления на компьютере. Однако у CRC-сканеров имеется недостаток, заметно снижающий их эффективность: они не могут определить вирус в новых файлах (в электронной почте, на дискетах, в файлах, восстанавливаемых из backup или при распаковке файлов из архива), поскольку в их БД отсутствует информация об этих файлах.

К числу CRC-сканеров относится широко распространенная в России программа ADInf (Advanced Diskinfoscope) и ревизор AVP Inspector. Вместе с ADInf применяется лечащий модуль ADInf Cure Module (ADInfExt), который использует собранную ранее информацию о файлах для их восстановления после поражения неизвестными вирусами. В состав ревизора AVP Inspector также входит лечащий модуль, способный удалять вирусы.

Программы-блокировщики реализуют метод антивирусного мониторинга. Антивирусные блокировщики — это резидентные программы, перехватывающие «вирусо-опасные» ситуации и сообщаящие об этом пользователю. К «вирусо-опасным» ситуациям относятся вызовы, которые характерны для вирусов в моменты их размножения (вызовы на открытие для записи в выполняемые файлы, запись в загрузочные секторы дисков или MBR винчестера, попытки программ остаться резидентно и т. п.).

При попытке какой-либо программы произвести указанные действия блокировщик посылает пользователю сообщение и предлагает запретить

соответствующее действие. К достоинствам блокировщиков относится их способность обнаруживать и останавливать вирус на самой ранней стадии его размножения, что бывает особенно полезно в случаях, когда регулярно появляется давно известный вирус. Однако они не «лечат» файлы и диски. Для уничтожения вирусов требуется применять другие программы, например фаги. К недостаткам блокировщиков можно отнести существование путей обхода их защиты и их «назойливость» (например, они постоянно выдают предупреждение о любой попытке копирования исполняемого файла).

Следует отметить, что созданы антивирусные блокировщики, выполненные в виде аппаратных компонентов компьютера. Наиболее распространенной является встроенная в BIOS защита от записи в MBR винчестера.

Программы-иммунизаторы — это программы, предотвращающие заражение файлов. Иммунизаторы делятся на два типа: иммунизаторы, сообщающие о заражении, и иммунизаторы, блокирующие заражение каким-либо типом вируса. Иммунизаторы первого типа обычно записываются в конец файлов и при запуске файла каждый раз проверяют его на изменение. У таких иммунизаторов имеется один серьезный недостаток — они не могут обнаружить заражение стелс-вирусом. Поэтому этот тип иммунизаторов практически не используются в настоящее время.

Иммунизатор второго типа защищает систему от поражения вирусом определенного вида. Он модифицирует программу или диск таким образом, чтобы это не отражалось на их работе, вирус при этом воспринимает их зараженными и поэтому не внедряется. Такой тип иммунизации не может быть универсальным, поскольку нельзя иммунизировать файлы от всех известных вирусов. Однако в качестве полумеры подобные иммунизаторы могут вполне надежно защитить компьютер от нового неизвестного вируса вплоть до того момента, когда он будет определяться антивирусными сканерами.



## 2. РЕАЛИЗАЦИЯ ПРОГРАММЫ

### 2.1. Используемое ПО

Операционная система: Microsoft Windows 10 Home.

Среда разработки: Code::Blocks 20.03.

Компилятор: GNU g++.

### 2.2. Описание функций

#### 1) Функция main.

Функция main является главной, т.е. при запуске программы функция получает управление. Исходный код функции main находится в файле scannersignature.c.

Объявление функции:

```
int main();
```

Тип функции: int.

Аргументы функции: у функции нет аргументов.

Возвращаемое значение:

- 0 – Функция отработала корректно;
- 1 – Ошибка при изменении языка;
- 2 – Ошибка при выводе сообщения в функции printf;
- 3 – Ошибка при считывании пути к антивирусной базе в функции scanf;
- 4 – Ошибка при передаче указателя на антивирусную базу в качестве первого аргумента в функции GetSignature;
- 5 – Ошибка при передаче пути к антивирусной базе в качестве второго аргумента в функции GetSignature;
- 6 – Ошибка при передаче массива для хранения сигнатуры из антивирусной базы в качестве третьего аргумента в функции GetSignature;

- 7 – Ошибка при передаче переменной для хранения смещения в качестве четвертого аргумента в функции GetSignature;
- 8 – Ошибка при открытии антивирусной базы в функции GetSignature;
- 9 – Ошибка при считывании информации из антивирусной базы в функции GetSignature;
- 10 – Ошибка при очищении потока в функции fflush;
- 11 - Ошибка при выводе сообщения в функции printf;
- 12 – Ошибка при считывании пути к антивирусной базе в функции scanf;
- 13 – Ошибка при передаче указателя на проверяемый файл в качестве первого аргумента в функции FileFormat;
- 14 – Ошибка при передаче пути к проверяемому файлу в качестве второго аргумента в функции FileFormat;
- 15 – Ошибка при передаче массива для хранения сигнатуры из антивирусной базы в качестве третьего аргумента в функции FileFormat;
- 16 – Ошибка при передаче переменной для хранения смещения в качестве четвертого аргумента в функции FileFormat;
- 17 – Ошибка при открытии проверяемого файла в функции FileFormat;
- 18 – Ошибка при считывании двух первых байтов из проверяемого файла в функции FileFormat;
- 19 – Ошибка при выводе сообщения в функции FileFormat;
- 20 - Ошибка при закрытии проверяемого файла в функции FileFormat;
- 21 – Ошибка при передаче проверяемого файла внутри функции FileFormat в функцию FileScanning в качестве первого аргумента;

- 22 – Ошибка при передаче массива для хранения сигнатуры из антивирусной базы внутри функции FileFormat в функцию FileScanning;
- 23 – Ошибка при передаче переменной для хранения смещения внутри функции FileFormat в функцию FileScanning;
- 24 – Ошибка при передаче пути к проверяемому файлу внутри функции FileFormat в функцию FileScanning;
- 25 – Ошибка при установке указателя положения в проверяемом файле на последний байт в функции FileScanning;
- 26 – Ошибка при взятии смещения последнего байта в функции FileScanning;
- 27 – Ошибка при выводе сообщения в функции FileScanning;
- 28 – Ошибка при установке указателя положения в проверяемом файле по заданному смещению в функции FileScanning;
- 29 – Ошибка при считывании сигнатуры из проверяемого файла в функции FileScanning;
- 30 – Ошибка при выводе сообщения в функции FileScanning;
- 31 – Ошибка при выводе сообщения в функции FileScanning;
- 32 - Ошибка при закрытии проверяемого файла в функции FileFormat.

## 2) Функция GetSignature.

Функция GetSignature открывает файл антивирусной базы и считывает из него сигнатуру и смещение. Исходный код функции GetSignature находится в файле scannersignature.c.

Объявление функции:

```
int GetSignature(FILE ** AB, char w1[], unsigned char sign[], size_t *
shift);
```

Тип функции: int.

Аргументы функции:

- AV – указатель на указатель на переменную, хранящую файл антивирусной базы; тип аргумента: FILE \*\*;
- w1 – указатель на первый элемент массива, хранящего путь к антивирусной базе; тип аргумента: char \*;
- sign – указатель на первый элемент массива, для хранения сигнатуры из антивирусной базы; тип аргумента: unsigned char \*;
- shift – указатель на переменную для хранения смещения; тип аргумента: size\_t \*.

Возвращаемое значение:

- 0 – Функция отработала корректно;
- 1 – Первый аргумент функции является некорректным;
- 2 – Второй аргумент функции является некорректным;
- 3 – Третий аргумент функции является некорректным;
- 4 – Четвертый аргумент функции является некорректным;
- 5 – Ошибка при открытии файла антивирусной базы;
- 6 – Ошибка при считывании сигнатуры и смещения из файла антивирусной базы;
- 7 – Ошибка при закрытии файла антивирусной базы.

### 3) Функция FileFormat.

Функция FileFormat открывает проверяемый файл и определяет его тип. Исходный код функции FileFormat находится в файле scannersignature.c.

Объявление функции:

```
int FileFormat(FILE ** FFS, char w2[], unsigned char sign[], size_t * shift);
```

Тип функции: int.

Аргументы функции:

- FFS – указатель на указатель на переменную, хранящую проверяемый файл; тип аргумента: FILE \*\*;
- w2 – указатель на первый элемент массива, хранящего путь к проверяемому файлу; тип аргумента: char \*;

- sign – указатель на первый элемент массива, для хранения сигнатуры из антивирусной базы; тип аргумента: unsigned char \*;
- shift – указатель на переменную для хранения смещения; тип аргумента: size\_t \*.

Возвращаемое значение:

- 0 – Функция отработала корректно;
- 1 – Первый аргумент функции является некорректным;
- 2 – Второй аргумент функции является некорректным;
- 3 – Третий аргумент функции является некорректным;
- 4 – Четвертый аргумент функции является некорректным;
- 5 – Ошибка при открытии проверяемого файла;
- 6 – Ошибка при считывании первых двух байтов из проверяемого файла;
- 7 – Ошибка при выводе сообщения в функции printf;
- 8 - Ошибка при закрытии проверяемого файла;
- 9 – Ошибка при передаче проверяемого файла в функцию FileScanning в качестве первого аргумента;
- 10 – Ошибка при передаче массива для хранения сигнатуры из антивирусной базы в функцию FileScanning;
- 11 – Ошибка при передаче переменной для хранения смещения в функцию FileScanning;
- 12 – Ошибка при передаче пути к проверяемому файлу в функцию FileScanning;
- 13 – Ошибка при установке указателя положения в проверяемом файле на последний байт в функции FileScanning;
- 14 – Ошибка при взятии смещения последнего байта в функции FileScanning;
- 15 – Ошибка при выводе сообщения в функции FileScanning;

- 16 – Ошибка при установке указателя положения в проверяемом файле по заданному смещению в функции FileScanning;
- 17 – Ошибка при считывании сигнатуры из проверяемого файла в функции FileScanning;
- 18 – Ошибка при выводе сообщения в функции FileScanning;
- 19 – Ошибка при выводе сообщения в функции FileScanning;
- 20 – Ошибка при закрытии проверяемого файла.

#### 4) Функция FileScanning.

Функция FileScanning сравнивает сигнатуру в проверяемом файле с сигнатурой из антивирусной базы данных по смещению из нее же. Исходный код функции FileScanning находится в файле scannersignature.c.

Объявление функции:

```
int FileScanning(FILE ** FFS, unsigned char sign[], size_t * shift, char w2[]);
```

Тип функции: int.

Аргументы функции:

- FFS – указатель на указатель на переменную, хранящую проверяемый файл; тип аргумента: FILE \*\*;
- sign – указатель на первый элемент массива, для хранения сигнатуры из антивирусной базы; тип аргумента: unsigned char \*;
- shift – указатель на переменную для хранения смещения; тип аргумента: size\_t \*;
- w2 – указатель на первый элемент массива, хранящего путь к проверяемому файлу; тип аргумента: char \*.

Возвращаемое значение:

- 0 – Функция отработала корректно;
- 1 – Первый аргумент функции является некорректным;
- 2 – Второй аргумент функции является некорректным;
- 3 – Третий аргумент функции является некорректным;

- 4 – Четвертый аргумент функции является некорректным;
- 5 – Ошибка при установке указателя положения в проверяемом файле на последний байт;
- 6 – Ошибка при взятии смещения последнего байта;
- 7 – Ошибка при выводе сообщения;
- 8 – Ошибка при установке указателя положения в проверяемом файле по заданному смещению;
- 9 – Ошибка при считывании сигнатуры из проверяемого файла;
- 10 – Ошибка при выводе сообщения;
- 11 – Ошибка при выводе сообщения.

### 3. РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

При запуске программа просит ввести путь к антивирусной базе. Сообщение в консоли продемонстрировано на рисунке 1.

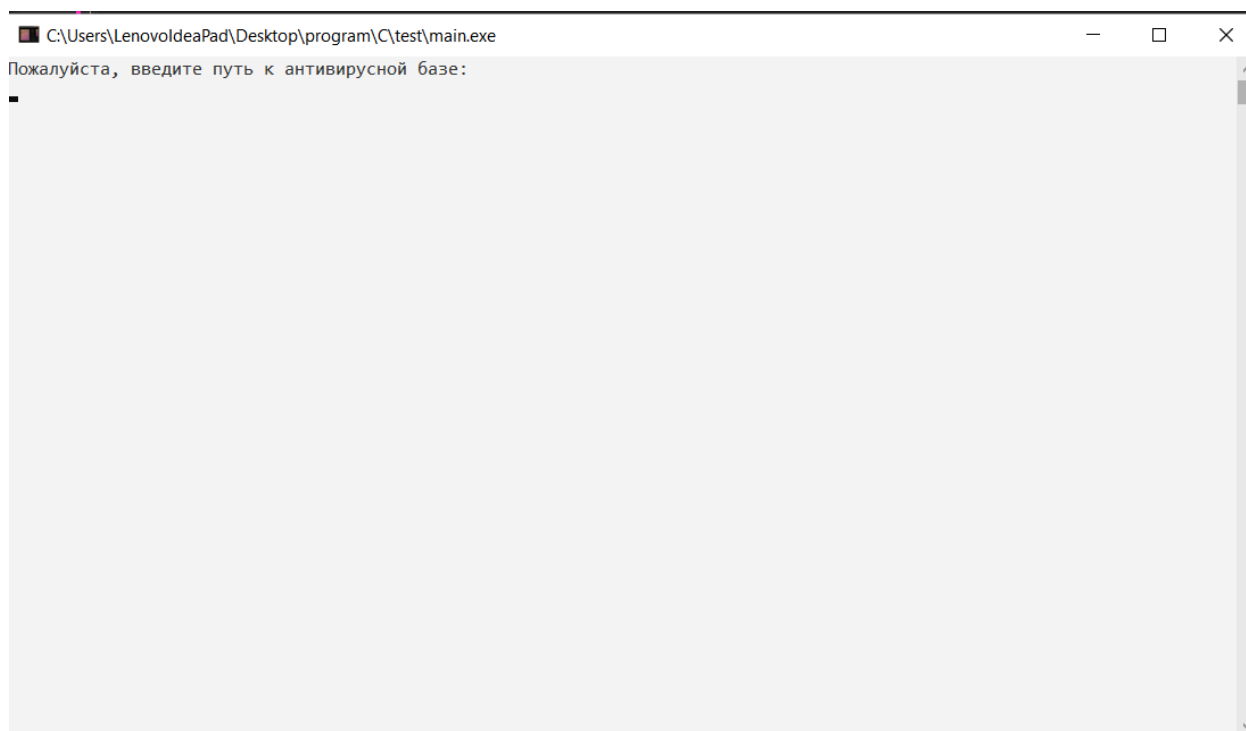


Рисунок 1 – Консольное окно при запуске программы

После ввода пути к антивирусной базе необходимо ввести путь к проверяемому файлу. Сообщение в консоли продемонстрировано на рисунке 2.



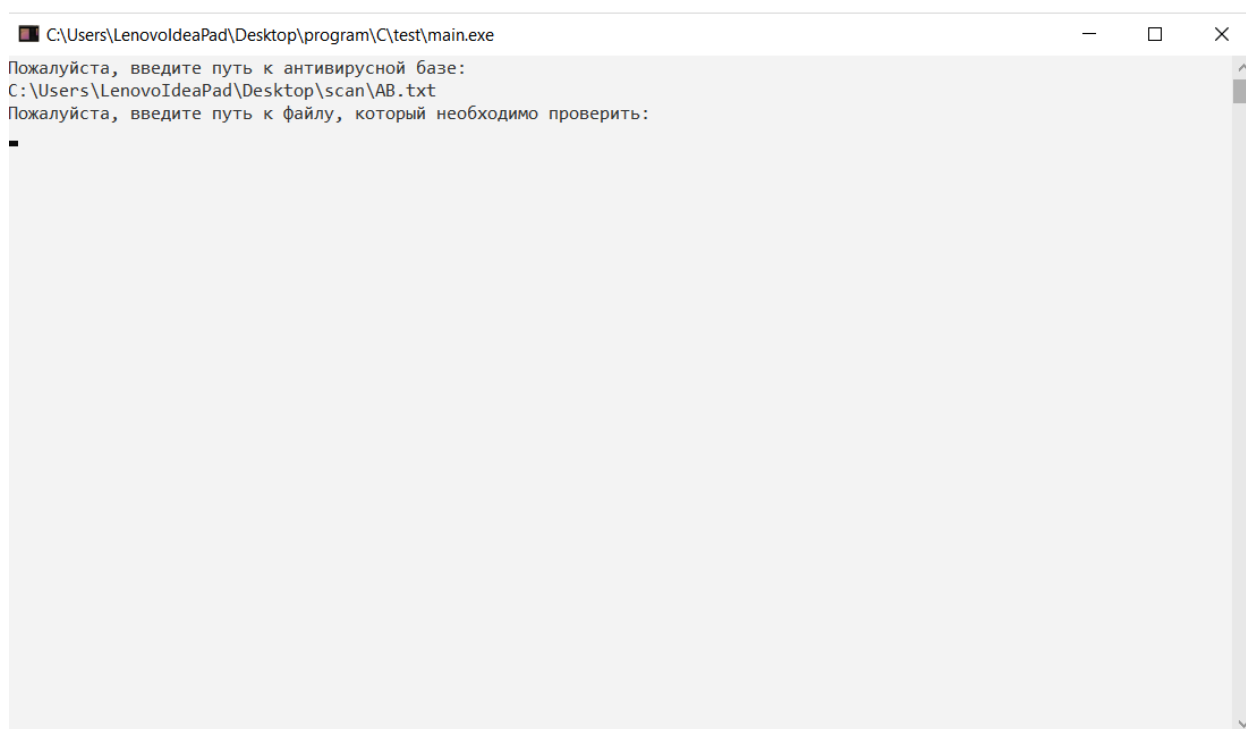
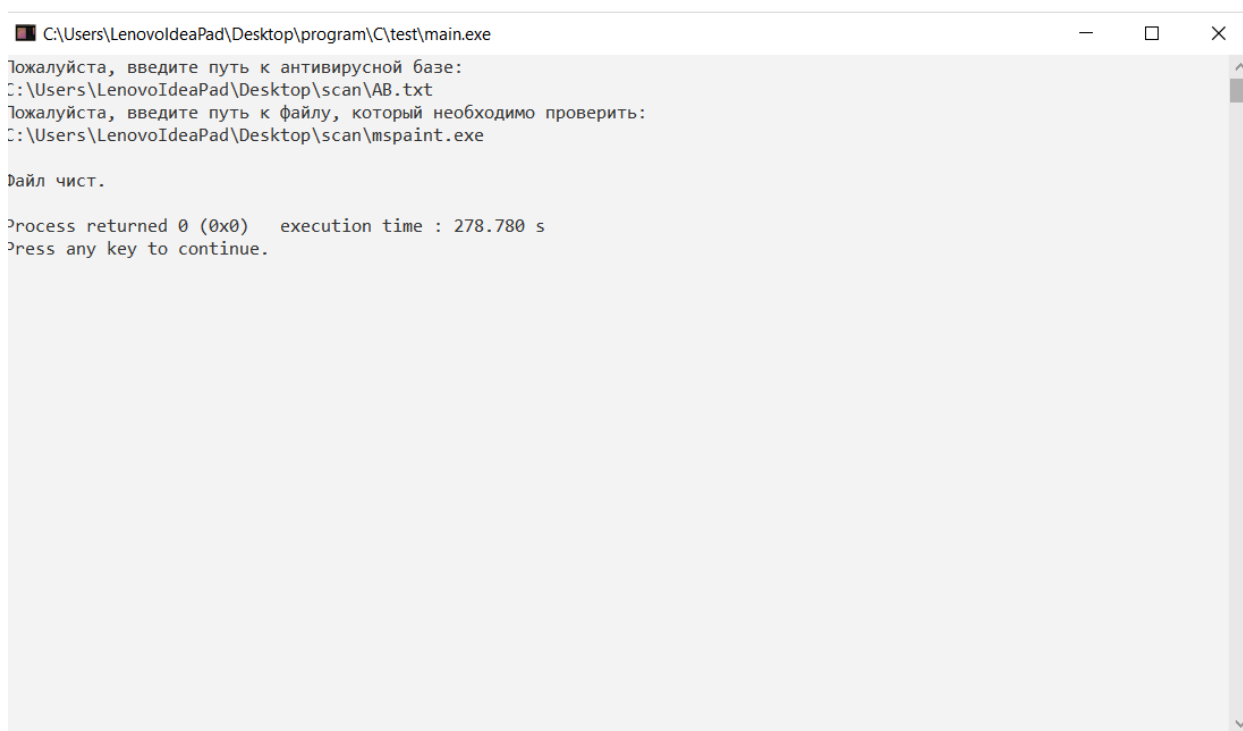


Рисунок 2 – Консольное окно при вводе пути к антивирусной базе

После ввода пути к проверяемому файлу программа выводит информацию о проверке. Если файл оказался чистым, выводится сообщение «Файл чист.» Сообщение в консоли продемонстрировано на рисунке 3.



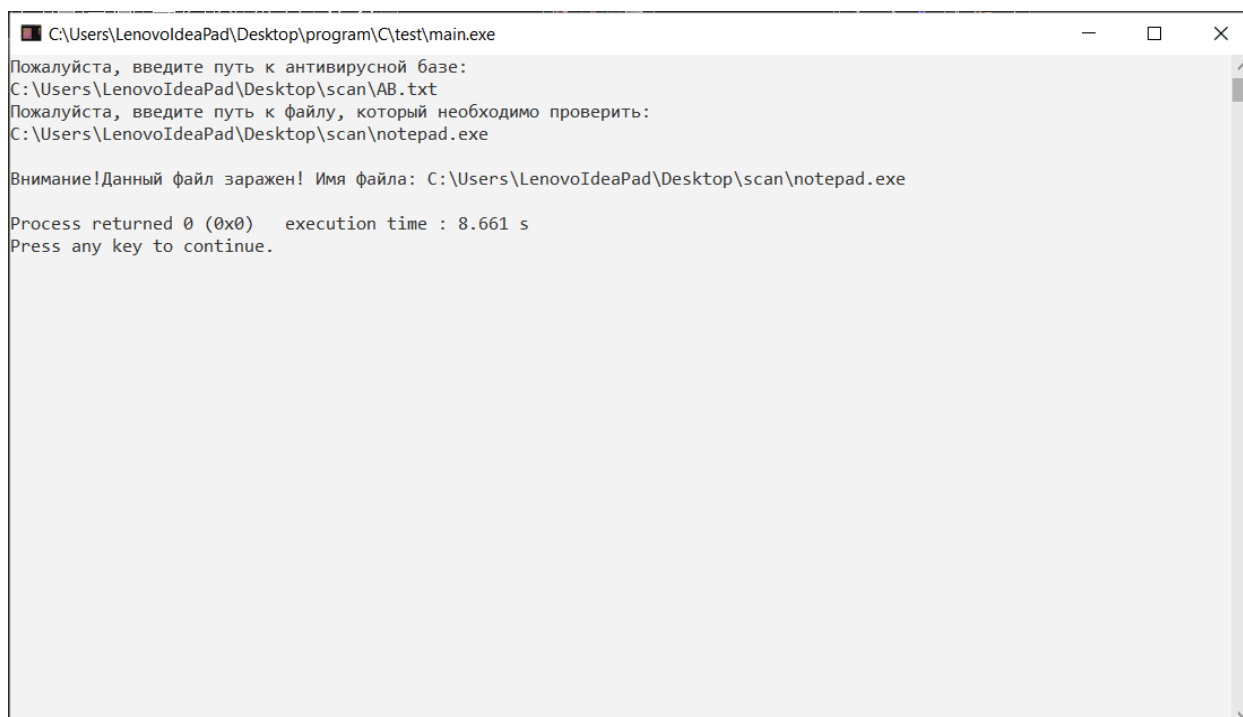
```
C:\Users\LenovoIdeaPad\Desktop\program\C\test\main.exe
Пожалуйста, введите путь к антивирусной базе:
C:\Users\LenovoIdeaPad\Desktop\scan\AB.txt
Пожалуйста, введите путь к файлу, который необходимо проверить:
C:\Users\LenovoIdeaPad\Desktop\scan\mspaint.exe

Файл чист.

Process returned 0 (0x0)   execution time : 278.780 s
Press any key to continue.
```

Рисунок 3 – Консольное окно при положительном результате проверки файла

Если же файл оказался зараженным, выводится сообщение, показанное на рисунке 4.



```
C:\Users\LenovoIdeaPad\Desktop\program\C\test\main.exe
Пожалуйста, введите путь к антивирусной базе:
C:\Users\LenovoIdeaPad\Desktop\scan\AB.txt
Пожалуйста, введите путь к файлу, который необходимо проверить:
C:\Users\LenovoIdeaPad\Desktop\scan\notepad.exe

Внимание! Данный файл заражен! Имя файла: C:\Users\LenovoIdeaPad\Desktop\scan\notepad.exe

Process returned 0 (0x0)   execution time : 8.661 s
Press any key to continue.
```

Рисунок 4 – Консольное окно при отрицательном результате проверки файла

## **ЗАКЛЮЧЕНИЕ**

В ходе выполнения данной курсовой работы мною было написано консольное приложение на языке программирования Си. Созданное приложение способно проверить любой файл на наличие в нем сигнатуры, сообщающей о вредоносности.

Курсовая работа, а также информация, почерпанная из дополнительных источников, расширила мой кругозор в области информационной безопасности.

В ходе написания кода я закрепила знания по работе с файлами в языке Си, используя основные функции, прописывая к ним проверки на ошибки, очередной раз потренировалась работать с указателями, улучшила навык построения логически структурированной программы.

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Бирюков, А.А. Информационная безопасность: защита и нападение / А.А. Бирюков. - М.: ДМК Пресс, 2013. - 474 с.
2. Шаньгин, В.Ф. Информационная безопасность компьютерных систем и сетей: Учебное пособие / В.Ф. Шаньгин. - М.: ИД ФОРУМ, НИЦ Инфра-М, 2013. - 416 с.

## **ПРИЛОЖЕНИЕ 1. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

### **1. Краткое описание программы**

Программа предназначена для проверки файла на содержание в нем вредоносной сигнатуры из антивирусной базы.

### **2. Минимальные системные требования**

- Операционная система: Microsoft Windows 10 Home, 64 – разрядная операционная система;
- Процессор: как минимум 1 ГГц или SoC;
- ОЗУ: 2 ГБ;
- Место на жестком диске: 20 ГБ;
- Видеоадаптер: DirectX версии не ниже 9 с драйвером WDDM 1.0;
- Дисплей: 800 x 600.

### **3. Процесс установки**

Скопируйте файл `scannersignature.exe` в выбранную директорию.

### **4. Процесс запуска программы и работы с ней**

Перед запуском программы убедитесь, что на компьютере присутствует файл с антивирусной базой данных формата `txt`. После чего запустите `scannersignature.exe` из выбранной директории. Появится консольное окно, изображенное на рисунке 5.

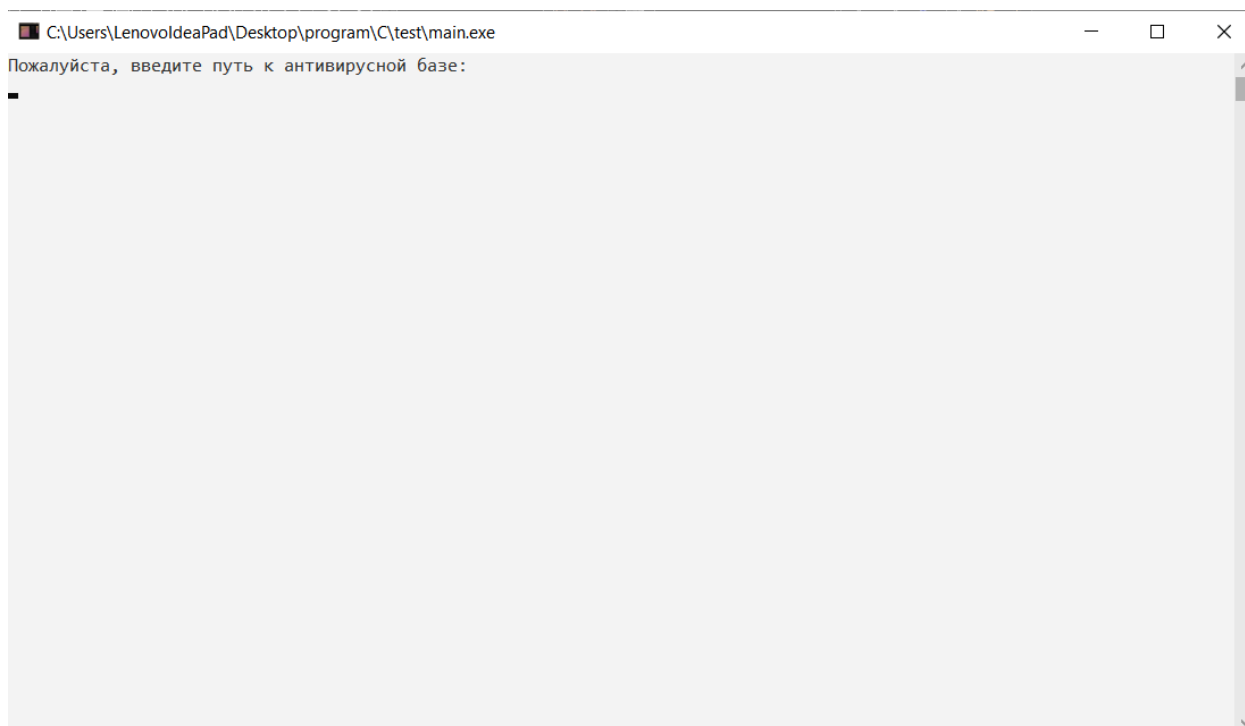


Рисунок 5 - Консольное окно при запуске программы

Программа попросит Вас ввести путь к антивирусной базе формата txt. Вводите путь так, как он указан в «Проводнике». Пример продемонстрирован на рисунке 6.

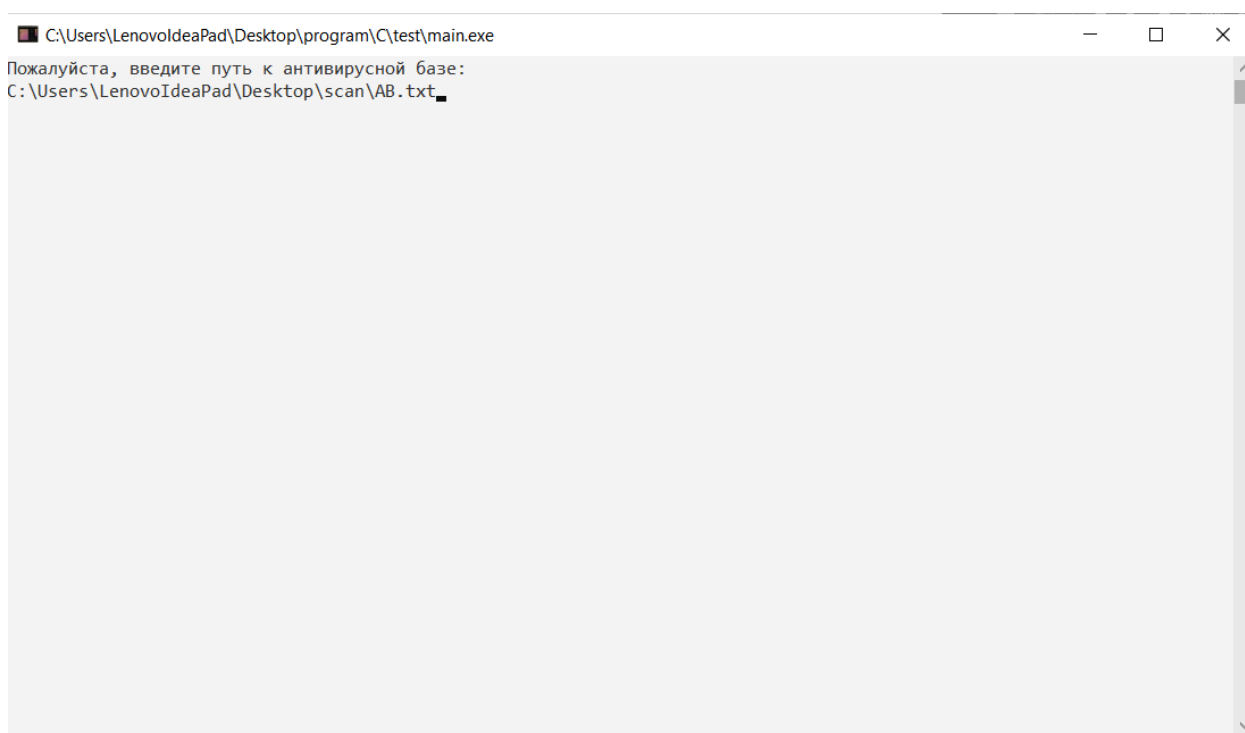


Рисунок 6 – Пример ввода пути к антивирусной базе

Далее программа попросит ввести путь к проверяемому файлу. Вводите путь аналогично пути к антивирусной базе. Пример продемонстрирован на рисунке 7.

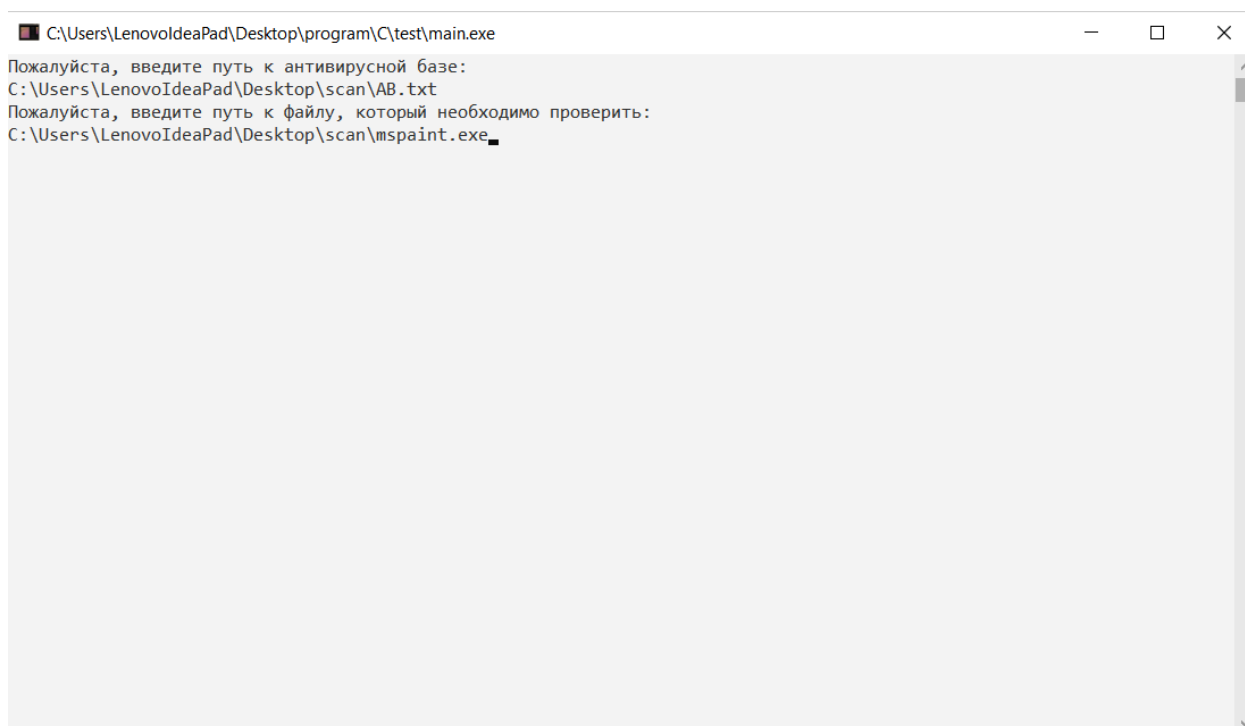


Рисунок 7 - Пример ввода пути к проверяемому файлу

После ввода пути к проверяемому файлу при соблюдении корректности в консоли отобразится результат проверки. Если файл чист, выведется соответствующее сообщение, изображенное на рисунке 8.

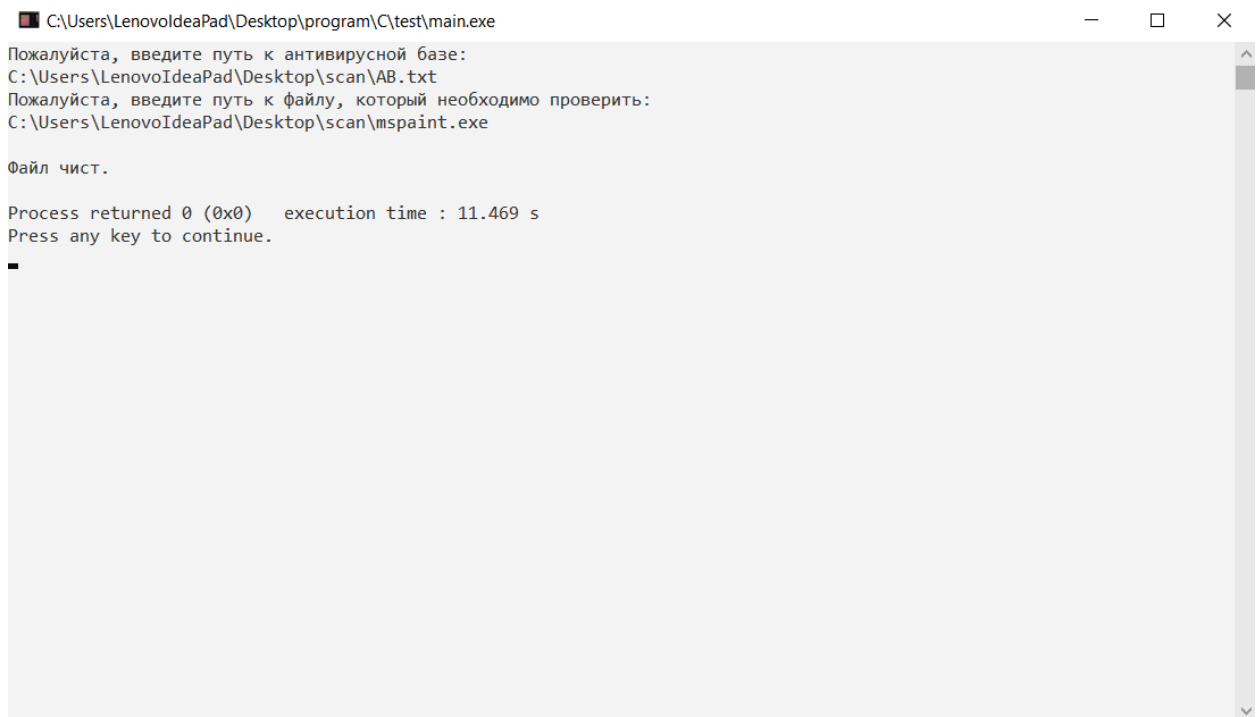


Рисунок 8 – Сообщение, если проверяемый файл не заражен

Иначе выведется сообщение, состоящее из информации о вредоносности файла и его пути. Отрицательный результат продемонстрирован на рисунке 9.

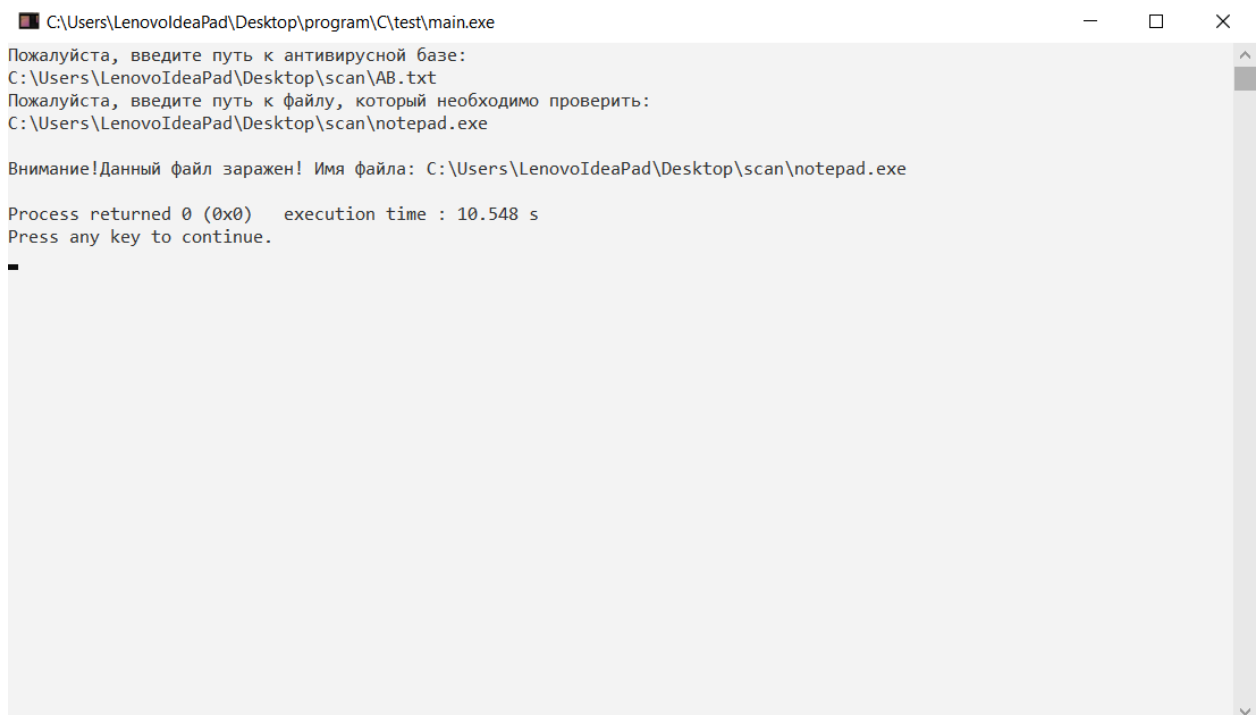
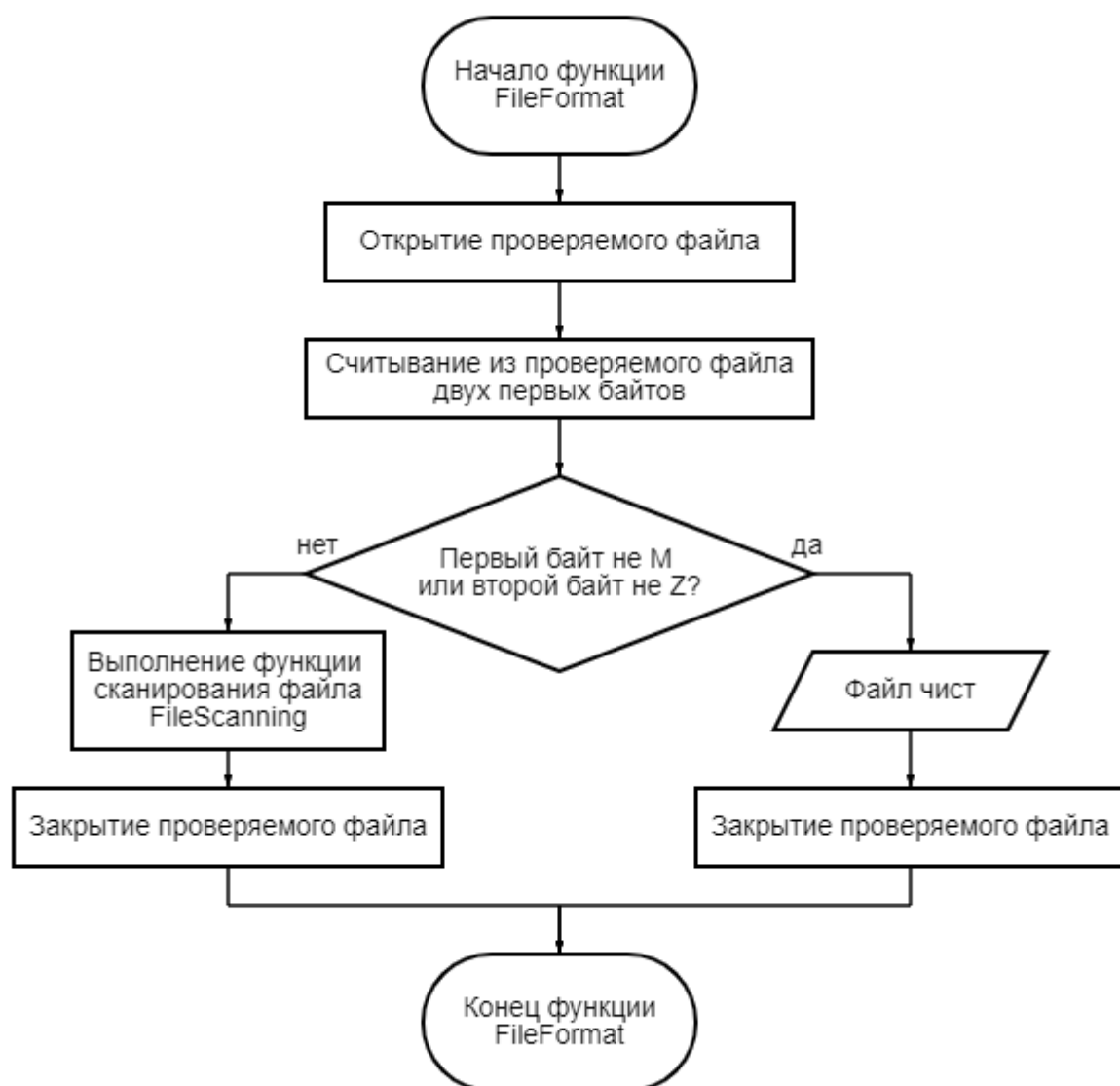


Рисунок 9 – Сообщение, если проверяемый файл заражен



Чтобы выйти из консольного окна, нажмите любую клавишу на клавиатуре.

## ПРИЛОЖЕНИЕ 2. БЛОК-СХЕМА АЛГОРИТМА



### ПРИЛОЖЕНИЕ 3. ИСХОДНЫЙ КОД

Файл scannersignature.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <stddef.h>
#include <string.h>
#include <windows.h>

int GetSignature(FILE **, char [], unsigned char [],
size_t *);
int FileFormat(FILE **, char [], unsigned char [],
size_t *);
int FileScanning(FILE **, unsigned char [], size_t *,
char []);

int main()
{
    FILE* AB;
    FILE* FFS;
    int prove;
    int result;
    char *result2;
    size_t shift;
    unsigned char sign[8];
    char way1[MAX_PATH], way2[MAX_PATH];

    result2 = setlocale(LC_ALL, "Rus");
    if (result2 == NULL)
    {
```

```

        printf("\nПроизошла ошибка при изменении
языка.\n");
        return 1;
    }
    result = printf("Пожалуйста, введите путь к
антивирусной базе: \n");
    if (result < 0)
    {
        printf("\nПроизошла ошибка при выводе
сообщения.\n");
        return 2;
    }
    result = scanf("%[^\n]s", &way1);
    if (result < 1)
    {
        printf("\nПроизошла ошибка при считывании пути
к антивирусной базе.\n");
        return 3;
    }
    prove = GetSignature(&AB, way1, sign, &shift);
    if (prove != 0)
    {
        if (prove == 1)
        {
            printf("\nПроизошла ошибка при передаче
указателя (на антивирусную базу)\n");
            return 4;
        }
        else if (prove == 2)
        {

```

```

        printf("\nПроизошла ошибка при передаче
пути к антивирусной базе.\n");
        return 5;
    }
    else if (prove == 3)
    {
        printf("\nПроизошла ошибка при передаче
массива для хранения сигнатуры из антивирусной
базы.\n");
        return 6;
    }
    else if (prove == 4)
    {
        printf("\nПроизошла ошибка при передаче
переменной для хранения смещения.\n");
        return 7;
    }
    else if (prove == 5)
    {
        printf("\nПроизошла ошибка при открытии
антивирусной базы.\n");
        return 8;
    }
    else if (prove == 6)
    {
        printf("\nПроизошла ошибка при
считывании информации из антивирусной базы.\n");
        return 9;
    }
}

```

```

    result = fflush(stdin);
    if (result != 0)
    {
        printf("\nПроизошла ошибка при очищении
входного потока.\n");
        return 10;
    }
    result = printf("Пожалуйста, введите путь к файлу,
который необходимо проверить: \n");
    if (result < 0)
    {
        printf("\nПроизошла ошибка при выводе
сообщения.\n");
        return 11;
    }
    result = scanf("%[^\n]s", &way2);
    if (result < 1)
    {
        printf("\nПроизошла ошибка при считывании пути
к файлу.\n");
        return 12;
    }
    prove = FileFormat(&FFS, way2, sign, &shift);
    if (prove != 0)
    {
        if (prove == 1)
        {
            printf("\nПроизошла ошибка при передаче
указателя на проверяемый файл в функцию

```

```

FileFormat.\n");
        return 13;
    }
    else if (prove == 2)
    {
        printf("\nПроизошла ошибка при передаче
пути к проверяемому файлу в функцию FileFormat.\n");
        return 14;
    }
    else if (prove == 3)
    {
        printf("\nПроизошла ошибка при передаче
массива для хранения сигнатуры из антивирусной базы в
функцию FileFormat.\n");
        return 15;
    }
    else if (prove == 4)
    {
        printf("\nПроизошла ошибка при передаче
переменной для хранения смещения в функцию
FileFormat.\n");
        return 16;
    }
    else if (prove == 5)
    {
        printf("\nПроизошла ошибка при открытии
проверяемого файла.\n");
        return 17;
    }
    else if (prove == 6)

```

```

        {
            printf("\nПроизошла ошибка при
считывании двух первых байтов из проверяемого
файла.\n");
            return 18;
        }
        else if (prove == 7)
        {
            printf("\nПроизошла ошибка при выводе
сообщения (функция FileFormat).\n");
            return 19;
        }
        else if (prove == 8)
        {
            printf("\nПроизошла ошибка при закрытии
проверяемого файла (функция FileFormat).\n");
            return 20;
        }
        else if (prove == 9)
        {
            printf("\nПроизошла ошибка при передаче
проверяемого файла в функцию FileScanning.\n");
            return 21;
        }
        else if (prove == 10)
        {
            printf("\nПроизошла ошибка при передаче
массива для хранения сигнатуры из антивирусной базы в
функцию FileScanning.\n");
            return 22;
        }
    }
}

```



```

    }
    else if (prove == 11)
    {
        printf("\nПроизошла ошибка при передаче
переменной для хранения смещения в функцию
FileScanning.\n");
        return 23;
    }
    else if (prove == 12)
    {
        printf("\nПроизошла ошибка при передаче
пути к проверяемому файлу в функцию FileScanning.\n");
        return 24;
    }
    else if (prove == 13)
    {
        printf("\nПроизошла ошибка при
установке указателя положения в проверяемом файле на "
                "последний байт.\n");
        return 25;
    }
    else if (prove == 14)
    {
        printf("\nПроизошла ошибка при взятии
смещения последнего байта из проверяемого файла.\n");
        return 26;
    }
    else if (prove == 15)
    {
        printf("\nПроизошла ошибка при выводе

```

```

сообщения (функция FileScanning).\n");
        return 27;
    }
    else if (prove == 16)
    {
        printf("\nПроизошла ошибка при
установке указателя положения в проверяемом файле по "
                "заданному смещению.\n");
        return 28;
    }
    else if (prove == 17)
    {
        printf("\nПроизошла ошибка при
считывании сигнатуры из проверяемого файла.\n");
        return 29;
    }
    else if (prove == 18)
    {
        printf("\nПроизошла ошибка при выводе
сообщения (функция FileScanning).\n");
        return 30;
    }
    else if (prove == 19)
    {
        printf("\nПроизошла ошибка при выводе
сообщения (функция FileScanning).\n");
        return 31;
    }
    else if (prove == 20)
    {

```

```

        printf("\nПроизошла ошибка при закрытии
проверяемого файла (функция FileFormat).\n");
        return 32;
    }
}
return 0;
}

```

```

int GetSignature(FILE ** AB, char w1[], unsigned char
sign[], size_t * shift)
{
    if (AB == NULL) return 1;
    if (w1 == NULL) return 2;
    if (sign == NULL) return 3;
    if (shift == NULL) return 4;

    int Res;
    * AB = fopen(w1, "r");
    if (* AB == NULL) return 5;
    Res = fscanf(* AB,
"%hhhd%hhhd%hhhd%hhhd%hhhd%hhhd%hhhd%zu",
                &sign[0], &sign[1], &sign[2],
&sign[3], &sign[4], &sign[5], &sign[6], &sign[7],
shift);
    if (Res != 9)
    {
        fclose(* AB);
        return 6;
    }
    Res = fclose(*AB);
}

```

```

    if (Res != 0)
    {
        return 7;
    }
    return 0;
}

int FileFormat(FILE ** FFS, char w2[], unsigned char
sign[], size_t * shift)
{
    if (FFS == NULL) return 1;
    if (w2 == NULL) return 2;
    if (sign == NULL) return 3;
    if (shift == NULL) return 4;

    int Res;
    unsigned char form[2];
    * FFS = fopen(w2, "rb");
    if (* FFS == NULL) return 5;

    Res = fread(form, sizeof(unsigned char), 2, * FFS);
    if (Res != 2)
    {
        fclose(* FFS);
        return 6;
    }
    if (form[0] != 'M' || form[1] != 'Z')
    {
        Res = printf("\nВсе хорошо, файл чист.\n");
        if (Res < 0) return 7;
    }
}

```

```

        Res = fclose(* FFS);
        if (Res != 0)
        {
            return 8;
        }
    }
else {
    Res = FileScanning(FFS, sign, shift, w2);
    if (Res != 0)
    {
        fclose(* FFS);
        return Res + 8;
    }
    Res = fclose(* FFS);
    if (Res != 0)
    {
        return 20;
    }
}
return 0;
}

int FileScanning(FILE ** FFS, unsigned char sign[],
size_t * shift, char w2[])
{
    if (FFS == NULL) return 1;
    if (sign == NULL) return 2;
    if (shift == NULL) return 3;
    if (w2 == NULL) return 4;

```

```

int Res;
size_t LastShift;
unsigned char list[8];

Res = fseek(* FFS, -1, SEEK_END);
if (Res != 0)
{
    fclose(* FFS);
    return 5;
}
LastShift = ftell(* FFS);
if (LastShift == -1L)
{
    fclose(* FFS);
    return 6;
}
if (LastShift - * shift < 7)
{
    Res = printf("\nФайл чист.\n");
    if (Res < 0) return 7;
    return 0;
}

Res = fseek(* FFS, * shift, SEEK_SET);
if (Res != 0)
{
    fclose(* FFS);
    return 8;
}
Res = fread(list, sizeof(unsigned char), 8, * FFS);

```

```

    if (Res != 8)
    {
        fclose(* FFS);
        return 9;
    }
    if (memcmp(list, sign, 8) == 0)
    {
        Res = printf("\nВнимание! Данный файл заражен!
Имя файла: %s\n", w2);
        if (Res < 0)
        {
            fclose(* FFS);
            return 10;
        }
    }
    else
    {
        Res = printf("\nФайл чист.\n");
        if (Res < 0)
        {
            fclose(* FFS);
            return 11;
        }
    }
    return 0;
}

```