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. 9362 _____ Красов К.С.

Преподаватель _____ Халиуллин Р.А.

Санкт-Петербург

2020

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Красов К.С.

Группа 9362

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. 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, принцип работы сигнатурных сканеров, анализ файла, реализация программы, выбранный язык программирования, выбранное программное обеспечение (ПО) для разработки, реализация функций, результаты тестирования программы, личный вклад, заключение, список использованных источников, приложение 1 – руководство пользователя, приложение 2 – блок-схема алгоритма, приложение 3 – исходный код программы.

Предполагаемый объём пояснительной записки:

Не менее 40 страниц.

Дата выдачи задания: 18.02.2020

Дата сдачи реферата: 03.06.2020

Дата защиты реферата: 09.06.2020

Студент _____ Красов К.С.

Преподаватель _____ Халиуллин Р.А.

АННОТАЦИЯ

Курсовая работа посвящена разработке приложения «сигнатурный сканер». Для этого был использован язык C++. Интерфейс — консольный, для управления используется клавиатура. В исходном коде реализованы проверки возвращаемых функциями (как сторонними, так и реализованными) значений, и обработка исключений. Приложение ищет определённую сигнатуру по определённому сдвигу байт в файлах формата Portable Executable. Приложение может использоваться как квалифицированными, так и неквалифицированными пользователями.

Ключевые слова: консольный интерфейс, сигнатура, сдвиг байт, формат PE.

SUMMARY

Course work is devoted to the development of the application «Signature scanner». The interface is console based, and the keyboard is used for control. The source code implements checks of values returned by functions (both third-party and implemented), and exception handling. The application searches for a specific signature based on a specific byte shift in files in the Portable Executable format. The app can be used by both qualified and unqualified users.

Keywords: command-line interface, the signature, the offset bytes, PE format.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Обнаружение сигнатур является одним из основных методов работы антивирусного программного обеспечения. Он основан на поиске уникальной последовательности байт в файле, которая характерна для определённого вируса.

С развитием общества всё более необходимым становится совершенствование способов защиты информации от несанкционированного доступа к ней. Одним из важнейших разделов информационной безопасности является компьютерная безопасность. За последние десятилетия появилось множество различного вредоносного программного обеспечения. Но с совершенствованием его, совершенствовались и методы обнаружения и борьбы с ним. Вредоносное программное обеспечение может наносить колоссальный ущерб, как отдельным лицам, так и определённой части общества.

Важно отметить, что создание или распространение программного обеспечения, которое предназначено для несанкционированного уничтожения, блокирования, модификации, копирования компьютерной информации или нейтрализации средств защиты компьютерной информации в Российской Федерации уголовно наказуемо.

Цель работы: Разработка приложения для поиска определённой сигнатуры в файлах.

Задачи:

- Изучить принцип работы сигнатурных сканеров;
- Выделить сигнатуру и сдвиг байт;
- Разработать алгоритм для поиска сигнатуры в файлах;
- Разработать приложение для поиска сигнатуры в файлах.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1. Принцип работы сигнатурных сканеров

Сигнатура – это уникальный набор свойств, который присущ определённому вирусу, благодаря которому возможно однозначно определить нахождение вируса в файле. Обычно в качестве сигнатуры выступает хэш, который генерируется на основе уже известного вредоносного программного обеспечения. Во множестве случаев уникальный набор свойств определяет не определённый вирус, а сразу целое семейство вирусов.

Разработчики антивирусного программного обеспечения составляют специальные базы сигнатур вредоносного программного обеспечения. Содержимое проверяемого файла сравнивается с базой. Если найдено совпадение, то такой файл считается заражённым. В базе, помимо сигнатур, хранятся также и способы лечения вирусов.

Сигнатурный анализ имеет ряд преимуществ. В первую очередь, этот метод позволяет выявить вирус и его тип с очень высокой долей вероятности и при этом исключая ложные срабатывания. И также сигнатурный анализ даёт возможность эффективно проводить лечение поражённых объектов.

Но сигнатурный анализ имеет и ряд недостатков. Выделение уникального набора свойств возможно только из уже известного вредоносного программного обеспечения. Поэтому сигнатурный анализ не позволяет обнаруживать новые, неизвестные ранее вирусы и их семейства. Также к недостаткам относится низкая скорость проверки. Не смотря на активное развитие аппаратного обеспечения и вследствие этого увеличения его производительности метод остаётся достаточно затратным по количеству используемых ресурсов. Разработчики антивирусного программного обеспечения различными способами сглаживают данный недостаток. Например, создаются технологии, которые позволяют не проверять файлы, которые уже подвергались проверке и, в которых с момента проверки не происходило изменений. Для этого антивирус ведёт специальную базу с

контрольными суммами объектов, проходивших проверку. И перед, непосредственно, проверкой файла по сигнатурам, сканер проверяет: изменился файл после последней проверки или нет. Также перед проверкой файла сначала устанавливается его формат. Например, в семействе операционных систем Microsoft Windows объект будет проверяться в том случае, если имеет формат Portable Executable (PE).

Формат PE – это формат исполняемых файлов всех Windows систем. Например, к нему относятся объекты с расширениями: «*.exe», «*.dll», «*.sys» и другие. Но достоверно определить формат файла по его расширению не представляется возможным, так как расширение возможно изменить, и таким образом замаскировать его. Формат возможно определить по первой структуре файла, которая носит название «IMAGE_DOS_HEADER» (DOS-заголовок). Содержимое структуры представлено на рисунке 1.

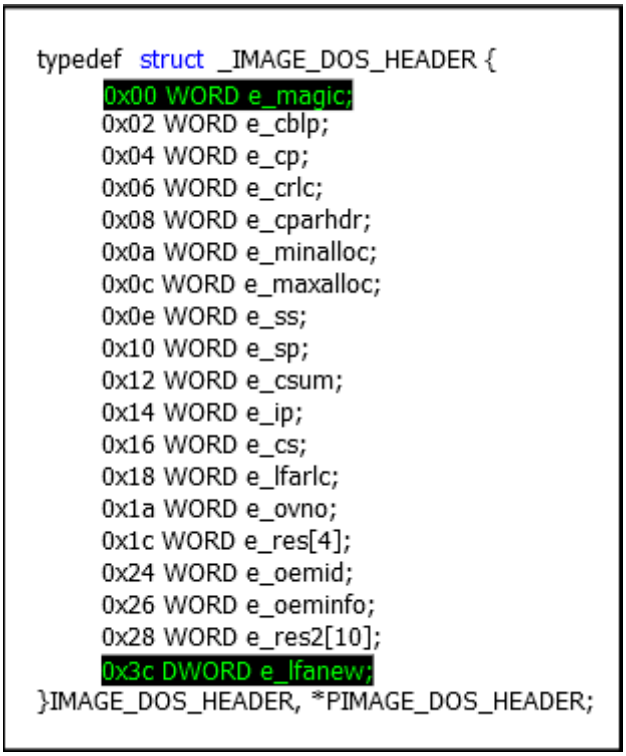


Рисунок 1 – Вид структуры «IMAGE_DOS_HEADER»

Для определения формата необходимо рассмотреть поле «e_magic». Это сигнатура, которая находится по сдвигу 0x0. Если она равна «MZ» (байты 0x4D и 0x5A), то данный файл имеет формат PE. На рисунке 2 приведено представление такого файла в hex-редакторе с выделенной сигнатурой «MZ».

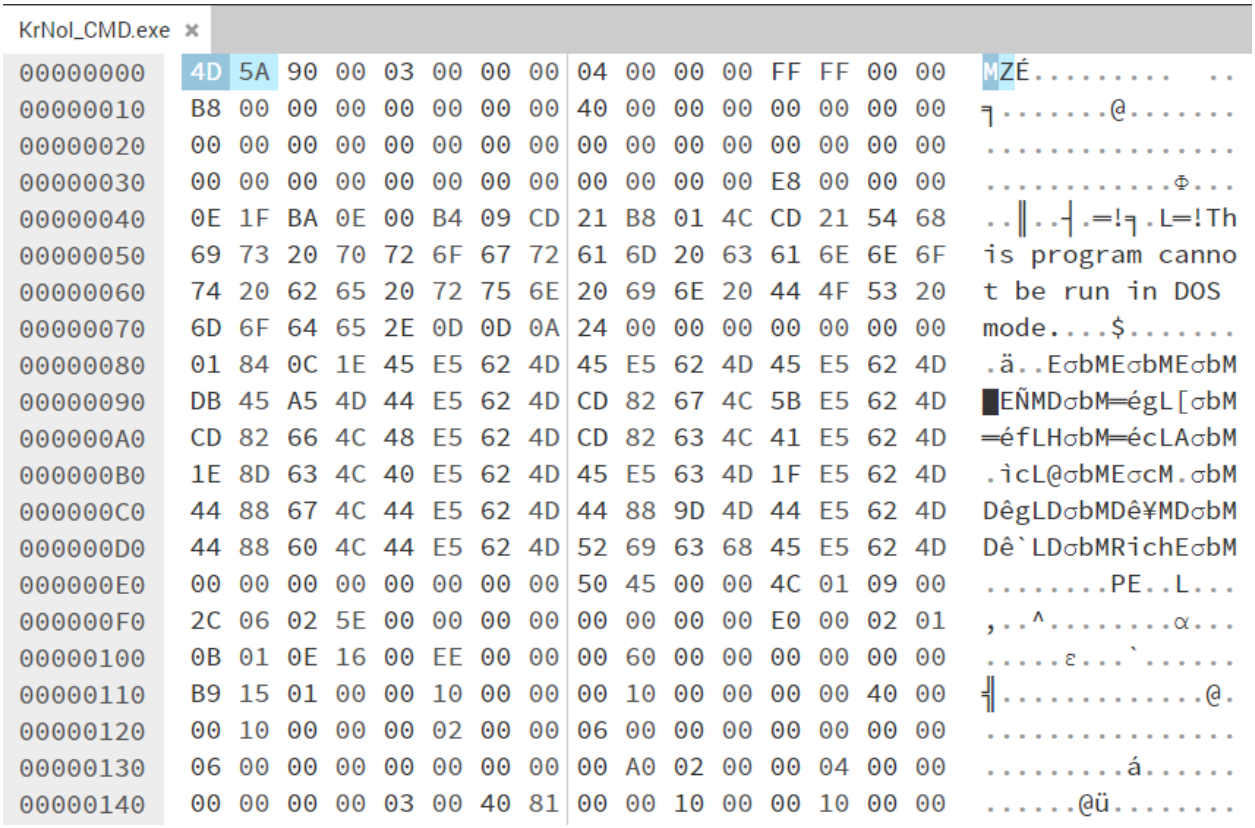


Рисунок 2 – Пример содержимого исполняемого файла

1.2. Анализ файла

Для сигнатурного анализа файла в первую очередь необходима сигнатура, которая определяет вирус или семейство вирусов, и её сдвиг. В основе анализа лежат следующие проверки:

- Проверка формата файла, которая необходима для того, чтобы убедиться, что он является исполняемым (так как если он таковым не является, то дальнейший анализ не имеет смысла);
- Сравнение участка исследуемого файла, который располагается в нём по заранее известному сдвигу с имеющейся сигнатурой.

Если участок идентичен сигнатуре, то данный файл является заражённым и в дальнейшем необходимо либо провести его лечение, либо удалить его. Иначе, при отсутствии совпадений, никаких действий не требуется, объект является безопасным.

2. РЕАЛИЗАЦИЯ ПРОГРАММЫ

2.1. Выбранный язык программирования

Для реализации приложения мною был выбран язык программирования C++.

2.2. Выбранное программное обеспечение (ПО) для разработки

Операционная система: Microsoft Windows Version 10.0.18363.476 (Microsoft Windows 10) разрядность x64.

Среда разработки (IDE): Microsoft Visual Studio 2019.

Компилятор: Microsoft Visual C++.

2.3 Реализация функций

Функция main.

Функция main – основная функция. Исходный код функции main находится в файле main.cpp.

Объявление функции:

int main();

Тип функции: int.

Аргументы функции:

- У функции нет аргументов.

Возвращаемое значение:

- 0 – программа завершилась без ошибок;
- 1 – ошибка в функции begin;
- 2 – ошибка в функции path_request;
- 3 – ошибка в функции folder_scan.

Функция begin.

Функция begin загружает данные, необходимые для дальнейшей работы программы. Исходный код функции begin находится в файле main.cpp.

Объявление функции:

int begin(int& ByteOffset, char*& Signat);

Тип функции: int.

Аргументы функции:

- ByteOffset – ссылка на целочисленную переменную, через которую выводится значение сдвига сигнатуры, тип аргумента: int&;
- Signat – ссылка на массив, через который выводится сигнатура, тип аргумента: char*&.

Возвращаемое значение:

- 0 – функция завершилась без ошибок;
- 3 – ошибка в функции setlocale;
- 4 – ошибка выделения памяти;
- 5 – ошибка в функции strcpy;
- 6 – ошибка чтения из файла;
- 7 – ошибка в функции close();
- 8 – ошибка в функции at;
- 9 – ошибка в функции at;
- 10 – ошибка в функции at;
- 11 – ошибка в строке SigStr (в строке есть элементы, которые не являются цифрами);
- 12 – ошибка в функции stoi;
- 13 – ошибка в функции open;
- 14 – ошибка в функции seekg;
- 15 – ошибка в функции tellg;
- 16 – ошибка в функции seekg;
- 17 – ошибка выделения памяти;
- 18 – ошибка в функции read;
- 19 – ошибка в функции close.

Функция path_request.

Функция path_request получает от пользователя адрес папки или файла. Исходный код функции path_request находится в файле main.cpp.

Объявление функции:

int path_request(std::string& adr);

Тип функции: int.

Аргументы функции:

- adr – ссылка на строку, через которую выводится информация об адресе, введённым пользователем, тип аргумента: std::string&.

Возвращаемое значение:

- 0 – функция завершилась без ошибок;
- 2 – ошибка в выводе в консоль;
- 3 – ошибка в функции exists;
- 4 – ошибка в выводе в консоль;
- 5 – ошибка в функции ignore.

Функция folder_scan.

Функция folder_scan осуществляет поочерёдное получение адресов для дальнейшего анализа файлов. Исходный код функции folder_scan находится в файле main.cpp.

Объявление функции:

int folder_scan(std::string& adr, int& ByteOffset, char*& Signat);

Тип функции: int.

Аргументы функции:

- adr – ссылка на строку, в которой хранится адрес каталога или файла, тип аргумента: std::string&;
- ByteOffset – ссылка на целочисленную переменную, в которой хранится значение сдвига сигнатуры, тип аргумента: int&;
- Signat – ссылка на массив, в котором хранится сигнатура, тип аргумента: char*&.

Возвращаемое значение:

- 0 – функция завершилась без ошибок;
- 1 – первый аргумент функции является некорректным;
- 2 – второй аргумент функции является некорректным;

- 3 – третий аргумент функции является некорректным;
- 4 – ошибка в функции is_directory;
- 5 – ошибка в конструкторе класса std::filesystem::recursive_directory_iterator;
- 6 – ошибка в конструкторе класса std::filesystem::directory_entry;
- 7 – ошибка в функции is_directory;
- 8 – ошибка в функции open;
- 9 – ошибка в функции read;
- 10 – ошибка в функции check_file;
- 11 – ошибка в функции close;
- 12 – ошибка в функции open;
- 13 – ошибка в функции read;
- 14 – ошибка в функции check_file;
- 15 – ошибка в выводе в консоль;
- 16 – ошибка в функции close.

Функция check_file.

Функция check_file проверяет наличие участка идентичного сигнатуре в проверяемом файле. Исходный код функции check_file находится в файле main.cpp.

Объявление функции:

int check_file(std::filesystem::directory_entry AdrFile, int& ByteOffset, char*& Signat);

Тип функции: int.

Аргументы функции:

- AdrFile – объект, который хранит адрес проверяемого файла, тип аргумента: std::filesystem::directory_entry;
- ByteOffset – ссылка на переменную, в которой хранится сдвиг сигнатуры, тип аргумента: int&;

- Signat – ссылка на массив, в котором хранится сигнатура, тип аргумента: char*&.

Возвращаемое значение:

- 0 – функция завершилась без ошибок;
- 1 – первый аргумент функции является некорректным;
- 2 – второй аргумент функции является некорректным;
- 3 – третий аргумент функции является некорректным;
- 4 – ошибка выделения памяти;
- 5 – ошибка в функции open;
- 6 – ошибка в функции seekg;
- 7 – ошибка в функции read;
- 8 – ошибка в функции close;
- 9 – ошибка вывода в консоль.

2.4 Результаты тестирования программы

При запуске приложения, оно сразу запрашивает у пользователя путь. Окно приложения при его запуске изображено на рисунке 3.

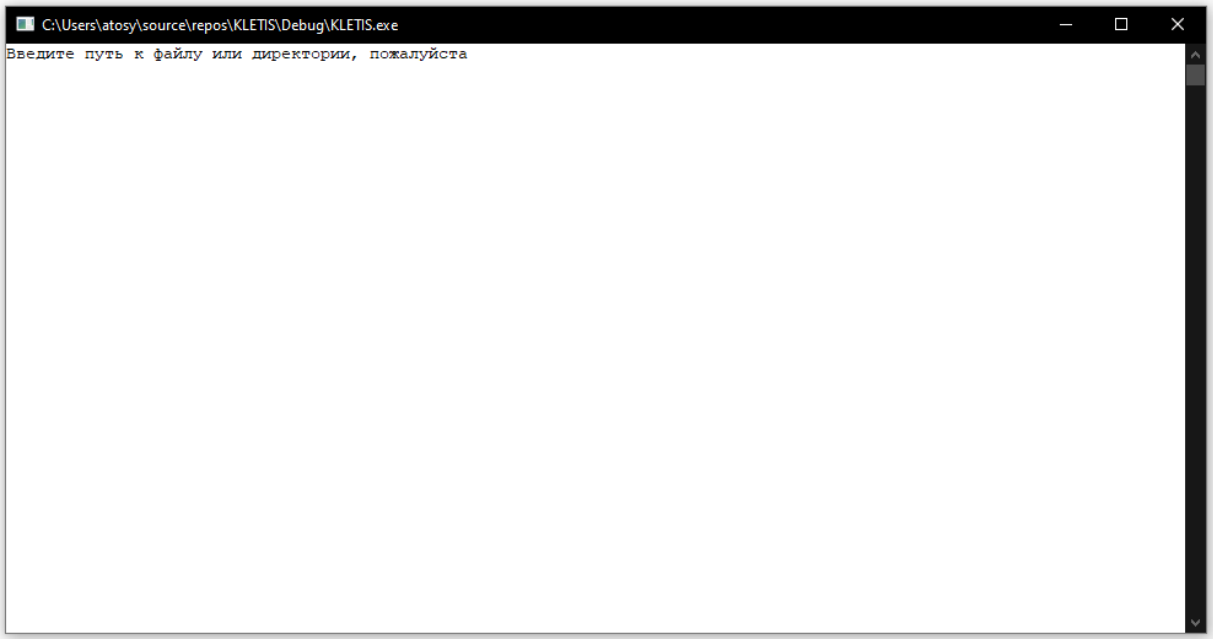


Рисунок 3 – Окно приложения при его запуске

Сообщение о некорректно введённом пути изображено на рисунке 4.

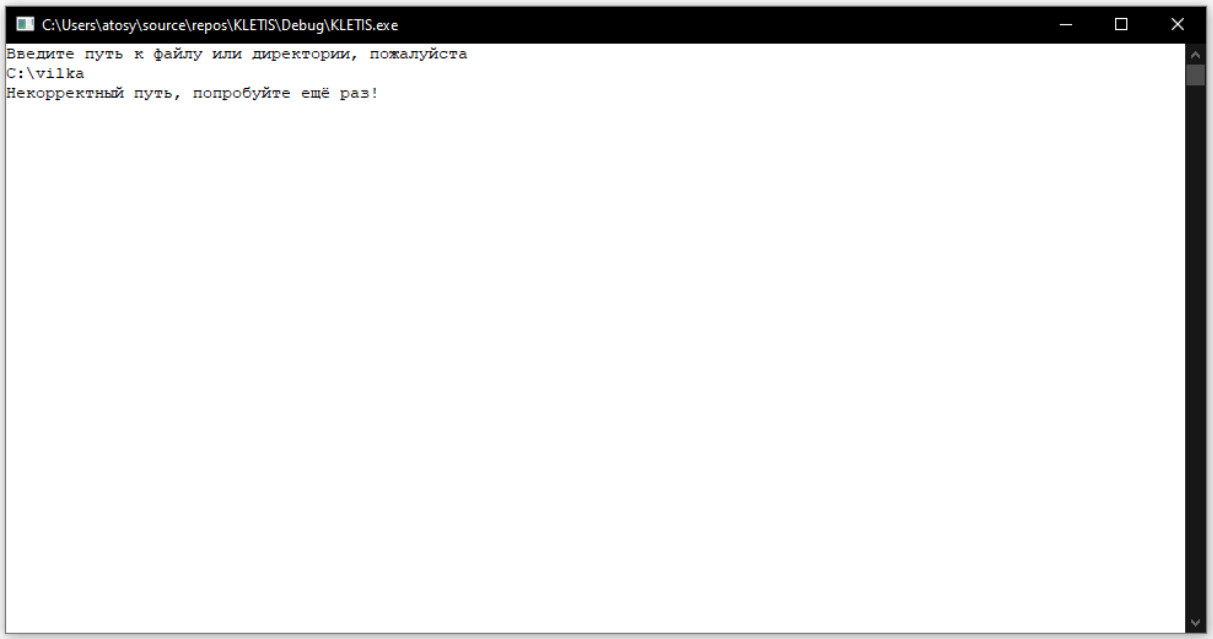


Рисунок 4 – Сообщение о некорректно введённом пути

Результат работы приложения, если в директории есть файлы, содержащие сигнатуру изображён на рисунке 5.

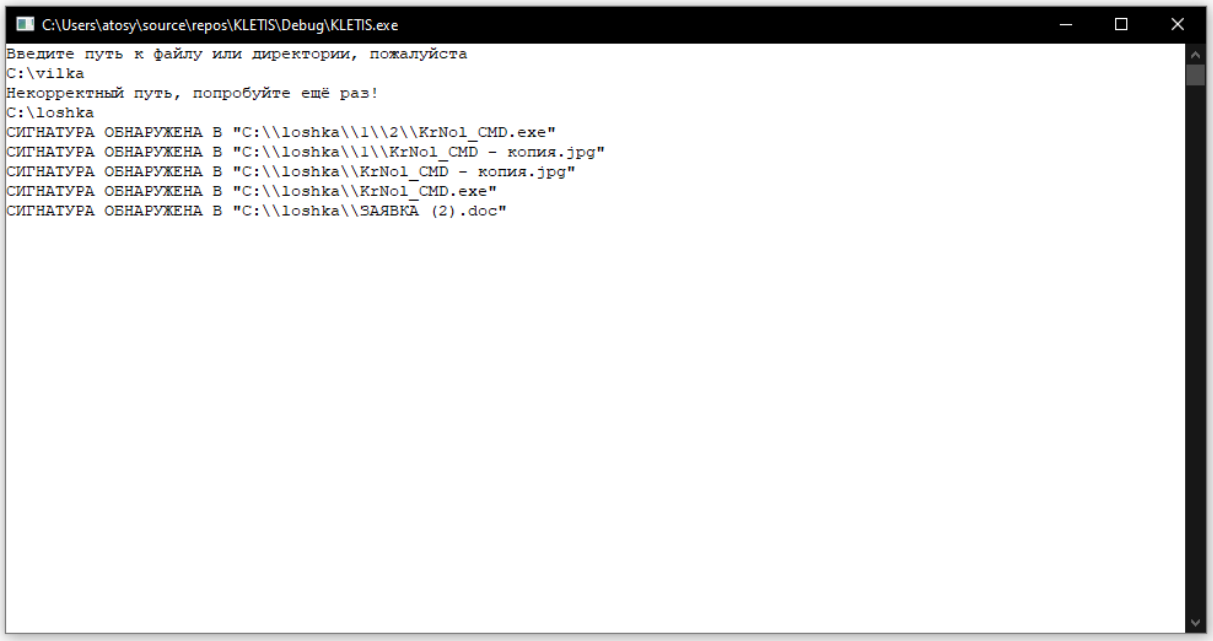


Рисунок 5 – Сообщения о файлах, содержащих сигнатуру

Результат работы приложения, если в директории нет файлов, содержащих сигнатуру изображён на рисунке 6.

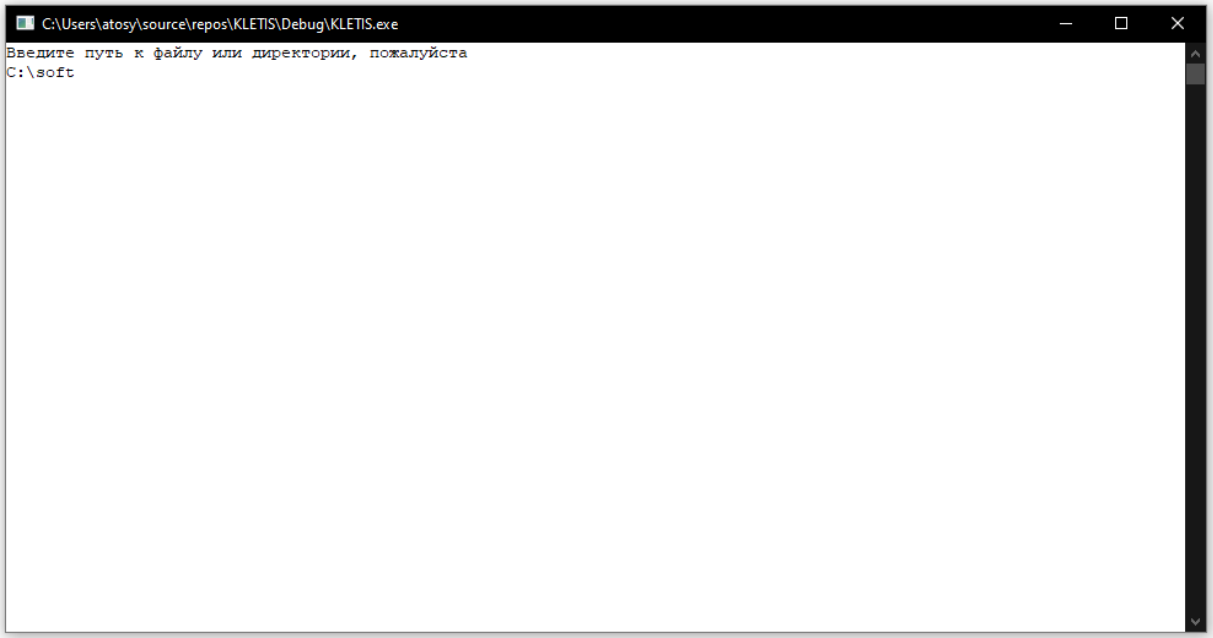


Рисунок 6 – Результат работы программы, при отсутсвии файлов, содержащих сигнатуру

Результат работы приложение при сканировании отдельного файла формата PE изображён на рисунке 7.

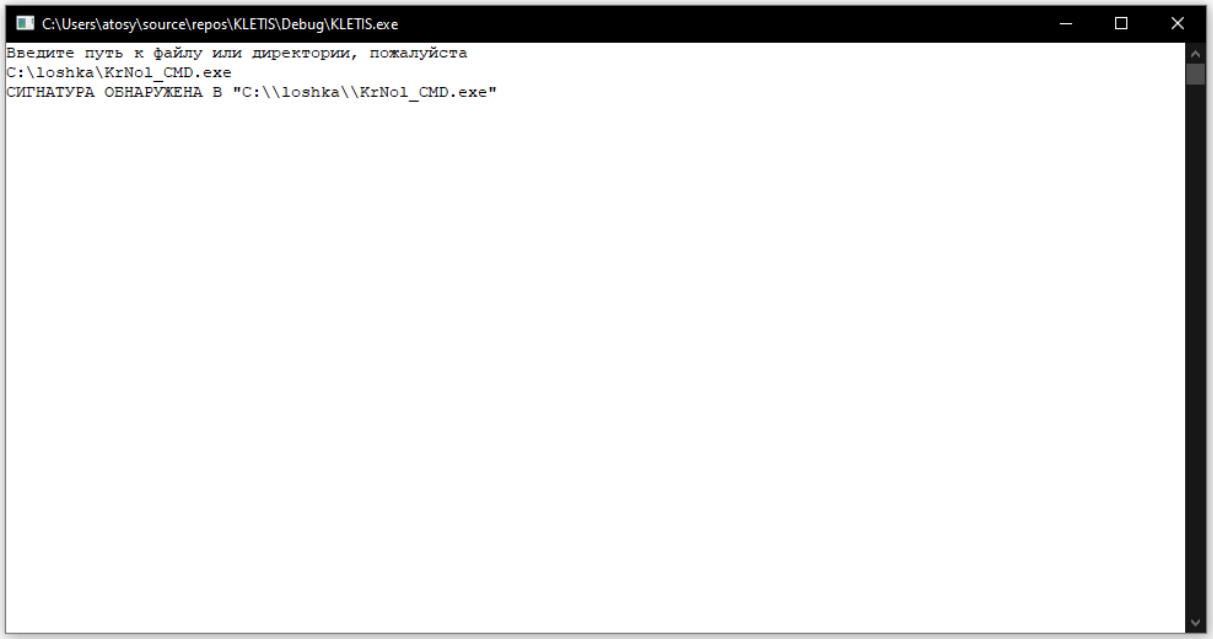


Рисунок 7 – Сканирование отдельного файла формата PE

Результат работы приложения при сканировании отдельного файла не формата PE изображён на рисунке 8.

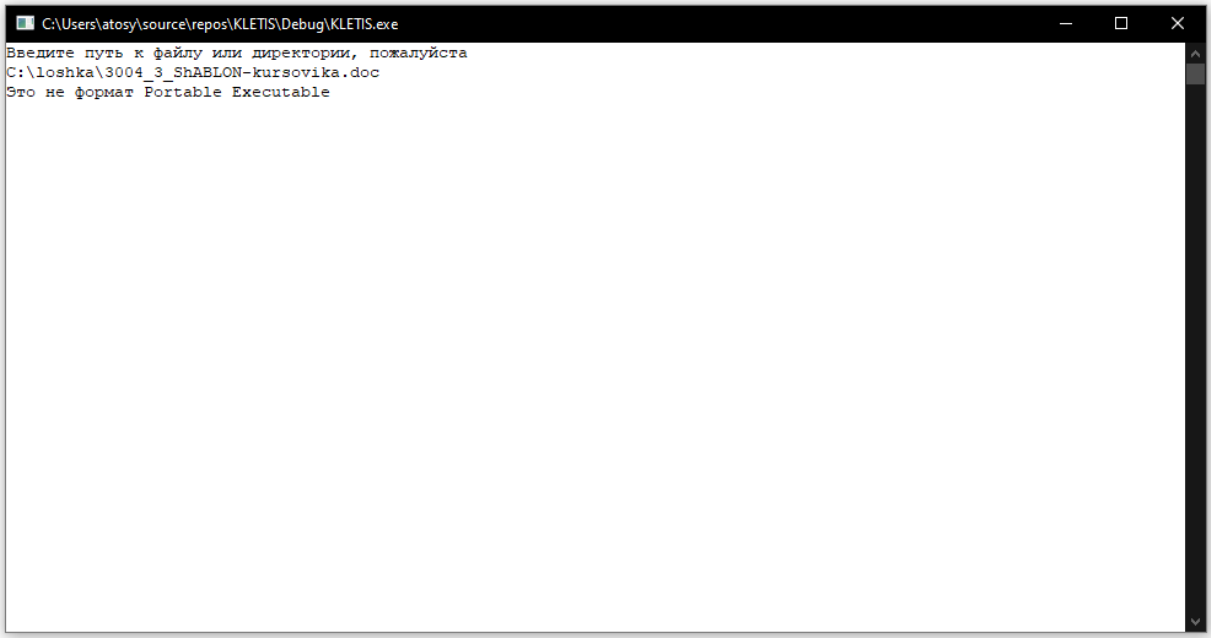


Рисунок 8 – Сканирование отдельного файла не формата PE

2.5 Личный вклад

Мною были реализованы функции: begin, folder_scan.

ЗАКЛЮЧЕНИЕ

Разработанное приложение «Сигнатурный сканер» реализовано на языке C++. Интерфейс – консольный. Возвращаемые значения функций проверяются, ошибки обрабатываются корректно.

Таким образом, беря во внимание всё вышеперечисленное, можно сделать вывод, что полученный результат соответствует поставленным цели и задачам.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 7.32-2017 «Межгосударственный стандарт. Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления»;
2. ГОСТ Р 7.0.97-2016 «Национальный стандарт Российской Федерации. Система стандартов по информации, библиотечному и издательскому делу. Организационно-распорядительная документация. Требования к оформлению документов»;
3. Требования к оформлению научно-технических отчетов (Распоряжение от 09.11.2015 № 3003);
4. ГОСТ 19.701-90 «Межгосударственный стандарт. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения»;
5. Керниган Б., Ритчи Д. «Язык программирования Си» М.: «Вильямс», 2017. 288 с;
6. Страуструп Б. «Язык программирования C++» М.: «Бином», 2015. 1136 с;
7. Лекция 6: Основы работы антивирусных программ. URL: https://www.intuit.ru/studies/courses/940/155/lecture/4304 (дата обращения: 07.06.2020);
8. Лекция 3: Структура программных компонентов. URL: https://www.intuit.ru/studies/courses/89/89/lecture/28301 (дата обращения: 07.06.2020);
9. Структура исполняемых файлов Win32 и Win64. URL: http://cs.usu.edu.ru/docs/pe/ (дата обращения: 07.06.2020).

ПРИЛОЖЕНИЕ 1
РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

I. Краткое описание приложения

Приложение создано для сигнатурного анализа файлов. Есть возможность проверять файлы в каталогах (включая подкаталоги), но также есть возможность проверять отдельные файлы. Интерфейс – консольный. Управление осуществляется при помощи клавиатуры.

II. Минимальные системные требования

Операционная система: Microsoft Windows 10 x64.

Процессор: Процессор с частотой 1 гигагерц (ГГц) или быстрее или система на кристалле SoC.

Видеоадаптер: DirectX 9 или более поздней версии с драйвером WDDM 1.0.

ОЗУ: 2 ГБ.

Дисплей: 800x600.

Интернет-соединение: не требуется.

III. Установка приложения

Скопируйте папку с приложением из архива в выбранную Вами директорию.

IV. Запуск приложения

Запустите файл KLETIS.exe (Дважды нажмите на файл KLETIS.exe).

V. Работа с программой

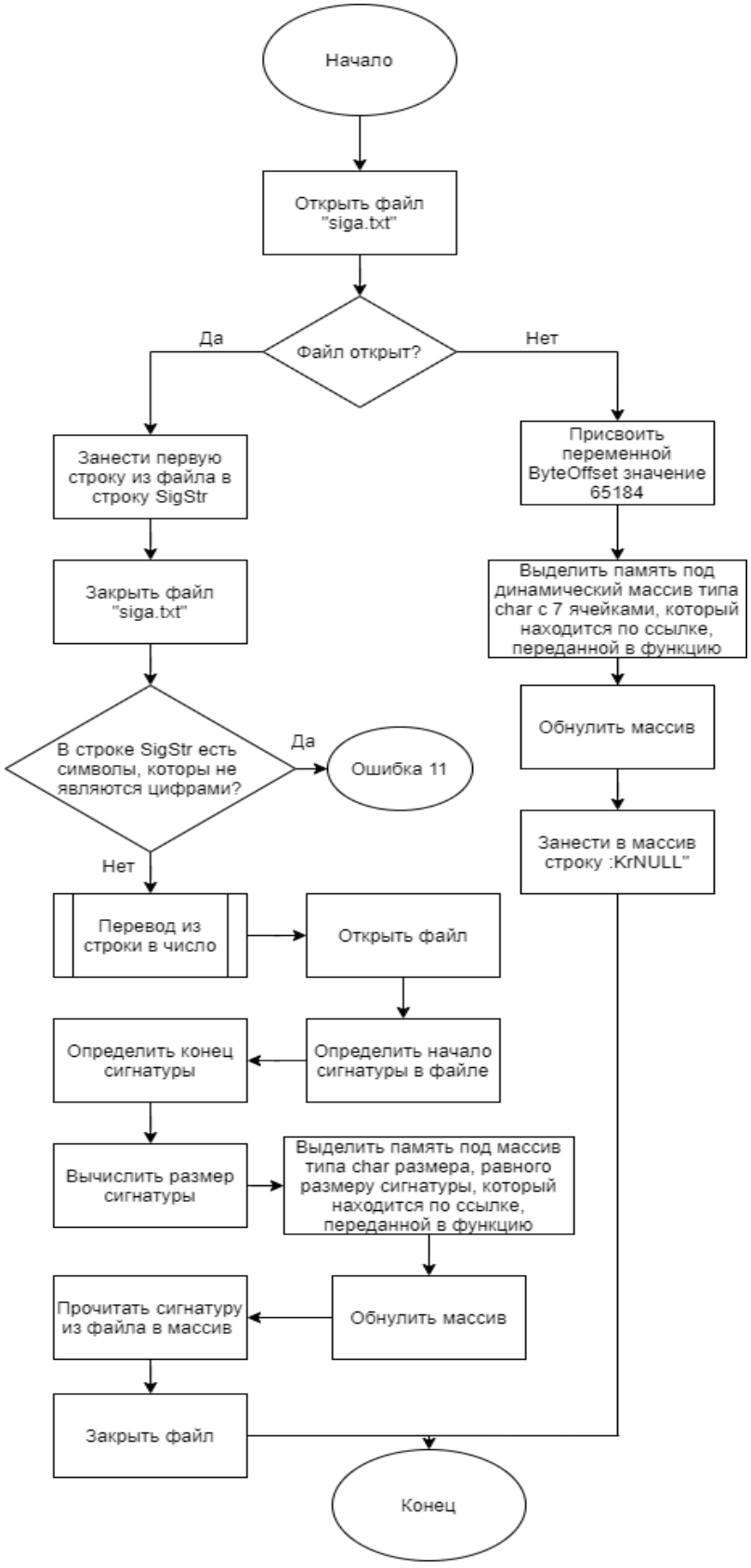
При запуске программа сразу запрашивает путь у Вас. Необходимо ввести путь к каталогу или путь к файлу. Для подтверждения ввода необходимо после того, как будет введён путь нажать клавишу «Enter». Если он был введён неверно, то программа сообщит об этом, и нужно будет ввести корректный путь (существующий путь). Название файлов и каталогов не должно содержать кириллических символов, так как в данном случае путь распознаётся, как некорректный.

Рассмотрим сначала работу программы, если введённый путь является каталогом. После того, как будет введён корректный путь, программа начнёт обрабатывать объекты по этому пути. Если заражённый файл (или файлы) будет найден, то программа выведет сообщение о том, что сигнатура обнаружена и выведет путь к файлу (или файлам, если таковых будет обнаружено несколько), в котором она обнаружена. Если заражённых файлов в каталоге не будет, то программа не выведет ничего. Важно отметить, что, помимо файлов в каталоге, программа также будет обрабатывать файлы во всех подкаталогах.

Теперь рассмотрим работу программу, если введённый путь является файлом. После того, как будет введён корректный путь, программа начнёт обрабатывать файл по этому пути. Если файл содержит сигнатуру, то программа сообщит об этом и выведет путь к файлу. Программа будет обрабатывать только тот файл, путь к которому был введён.

Файл, в котором находятся сдвиг и сигнатура, располагается в том же каталоге, что и сама программа. При отсутствии этого файла, программа будет использовать сдвиг и сигнатуру по умолчанию.

В файле с сигнатурой в первой строке находится значение сдвига. Начиная со второй строки и до конца файла находится сигнатура. Поведение программы в случае некорректного изменения файла с сигнатурой является неопределённым.



					Приложение 2. Блок-схема
Изм.	Лист	№ докум.	Подпись	Дата	
Разраб.		Красов К.С.			
Провер.		Халиуллин Р.А.			
Реценз.		Халиуллин Р.А.			
Н. Контр.		Халиуллин Р.А.			
Утверд.		Халиуллин Р.А.			

Лит.	Лист	Листов
	1	1

СПбГЭТУ «ЛЭТИ»

ПРИЛОЖЕНИЕ 3
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp:

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <math.h>
#include <Windows.h>
#include <sstream>
#include <string>
#include <fstream>
#include <filesystem>
#include <climits>

int begin(int& ByteOffset, char*& Signat);
int path_request(std::string& adr);
int folder_scan(std::string& adr, int& ByteOffset,
char*& Signat);
int check_file(std::filesystem::directory_entry AdrFile,
int& ByteOffset, char*& Signat);

int main()
{
    int ChIErr = 0;
    std::string adr;
    int ByteOffset = 0;
    char* Signat;
```

```
    ChIErr = begin(ByteOffset,Signat);
    if (ChIErr > 0)
    {
        std::cerr << "Error 1\n";
        return 1;
    }
    ChIErr = path_request(adr);
    if (ChIErr > 0)
    {
        std::cerr << "Error 2\n";
        delete[] Signat;
        return 2;
    }
    ChIErr = folder_scan(adr, ByteOffset, Signat);
    if (ChIErr > 0)
    {
        std::cerr << "Error 3\n";
        delete[] Signat;
        return 3;
    }
    delete[] Signat;
    return 0;
}

int begin(int& ByteOffset, char*& Signat)
{
    char* ChCPErr;
    int ChIErr = 0;
    char ChCErr = 0;
    size_t bn = 0;
```

```
size_t en = 0;
std::string SigStr;
ChCPErr = setlocale(LC_ALL, "");
if (ChCPErr == NULL)
{
    std::cerr << "Error 3\n";
    return 3;
}
std::ifstream SigFile("siga.txt");
if (!SigFile.is_open())
{
    ByteOffset = 0x000FEA0;
    try
    {
        Signat = new char[7];
    }
    catch (...)
    {
        std::cerr << "Error 4\n";
        return 4;
    }
    for (size_t i = 0; i < 7; ++i) Signat[i] = 0;
    ChCPErr = strcpy(Signat, "KrNULL");
    if (ChCPErr != Signat)
    {
        std::cerr << "Error 5\n";
        delete[] Signat;
        return 5;
    }
}
```

```
else
{
    try
    {
        SigFile >> SigStr;
    }
    catch (...)
    {
        std::cerr << "Error 6\n";
        SigFile.close();
        return 6;
    }
    try
    {
        SigFile.close();
    }
    catch (...)
    {
        std::cerr << "Error 7\n";
        return 7;
    }
    for (size_t i = 0; i < SigStr.size(); ++i)
    {
        try
        {
            ChCErr = SigStr.at(i);
        }
        catch (...)
        {
            std::cerr << "Error 8\n";
```

```
        return 8;
    }
    if ((ChCErr == '0') && (i == 0))
    {
        ++i;
        try
        {
            ChCErr = SigStr.at(i);
        }
        catch (...)
        {
            std::cerr << "Error 9\n";
            return 9;
        }
        if ((ChCErr == 'x') || (ChCErr == 'X'))
        {

            i = 2;
            try
            {
                ChCErr = SigStr.at(i);
            }
            catch (...)
            {
                std::cerr << "Error 10\n";
                return 10;
            }
        }
    }
```

```
            if (!(((ChCErr >= '0') && (ChCErr <= '9'))
|| ((ChCErr >= 'a') && (ChCErr <= 'f')) || ((ChCErr >=
'A') && (ChCErr <= 'F'))))
            {
                std::cerr << "Error 11\n";
                return 11;
            }
        }
        try
        {
            ByteOffset = std::stoi(SigStr, NULL, 16);
        }
        catch (...)
        {
            std::cerr << "Error 12\n";
            return 12;
        }
        try
        {
            SigFile.open("siga.txt",
std::ios_base::binary);
        }
        catch (...)
        {
            std::cerr << "Error 13\n";
            return 13;
        }
        bn = SigStr.size() + (size_t)2;
        try
        {
```

```
    SigFile.seekg(0, std::ios_base::end);
}
catch (...)
{
    std::cerr << "Error 14\n";
    SigFile.close();
    return 14;
}
try
{
    en = SigFile.tellg();
}
catch (...)
{
    std::cerr << "Error 15\n";
    SigFile.close();
    return 15;
}
try
{
    SigFile.seekg(bn, std::ios_base::beg);
}
catch (...)
{
    std::cerr << "Error 16\n";
    SigFile.close();
    return 16;
}
bn = en - bn;
try
```

```
        {
            Signat = new char[bn + 1];
        }
        catch (...)
        {
            std::cerr << "Error 17\n";
            SigFile.close();
            return 17;
        }
        for (size_t i = 0; i < bn + 1; ++i) Signat[i] =
0;
        try
        {
            SigFile.read(Signat, bn);
        }
        catch (...)
        {
            std::cerr << "Error 18\n";
            SigFile.close();
            delete[] Signat;
            return 18;
        }
        try
        {
            SigFile.close();
        }
        catch (...)
        {
            std::cerr << "Error 19\n";
            delete[] Signat;
```

```
            return 19;
        }
    }
    return 0;
}

int path_request(std::string& adr)
{
    bool ChBErr = 0;
    try
    {
        std::cout << "Введите путь к файлу или директории, пожалуйста\n";
    }
    catch (...)
    {
        std::cerr << "Error 2\n";
        return 2;
    }
    do
    {
        std::cin >> adr;
        try
        {
            ChBErr = std::filesystem::exists(adr);
        }
        catch (...)
        {
            std::cerr << "Error 3\n";
            return 3;
```

```
        }
        if (!ChBErr)
        {
            try
            {
                std::cout << "Некорректный путь,
попробуйте ещё раз!\n";
            }
            catch (...)
            {
                std::cerr << "Error 4\n";
                return 4;
            }
        }
        try
        {
            std::cin.ignore();
        }
        catch (...)
        {
            std::cerr << "Error 5\n";
            return 5;
        }
    } while (!ChBErr);
    return 0;
}

int folder_scan(std::string& adr, int& ByteOffset,
char*& Signat)
{
```

```
int ChIErr = 0;
bool ChBErr = 0;
size_t ChStErr = 0;
char dva[2] = { 0 };
try
{
    ChBErr = std::filesystem::exists(adr);
    if (!ChBErr) throw 1;
}
catch (...)
{
    std::cerr << "Error 1\n";
    return 1;
}
if (ByteOffset < 0)
{
    std::cerr << "Error 2\n";
    return 2;
}
if (Signat==NULL)
{
    std::cerr << "Error 3\n";
    return 3;
}
try
{
    ChBErr = std::filesystem::is_directory(adr);
}
catch (...)
{
```

```
            std::cerr << "Error 4\n";
            return 4;
        }
        if (ChBErr)
        {
            try
            {
                for (const auto& EntryFile :
std::filesystem::recursive_directory_iterator(adr))
                {
                    try
                    {
                        std::filesystem::directory_entry
AdrFile = EntryFile;
                        try
                        {
                            ChBErr =
AdrFile.is_directory();
                        }
                        catch (...)
                        {
                            std::cerr << "Error 7\n";
                            return 7;
                        }
                        if (!ChBErr)
                        {
                            std::ifstream FileChck;
                            try
                            {
```

```
    FileChck.open(AdrFile.path(),
std::ios_base::binary);
                        }
                        catch (...)
                        {
                            std::cerr << "Error 8\n";
                            return 8;
                        }
                        try
                        {
                            FileChck.read(dva, 2);
                        }
                        catch (...)
                        {
                            std::cerr << "Error 9\n";
                            FileChck.close();
                            return 9;
                        }
                        if ((dva[0] == 0x0000004D) &&
(dva[1] == 0x0000005A))
                        {
                            ChIErr                    =
check_file(AdrFile, ByteOffset, Signat);
                            if (ChIErr > 0)
                            {
                                std::cerr   <<   "Error
10\n";
                                FileChck.close();
                                return 10;
```

```
                    }
                }
                try
                {
                    FileChck.close();
                }
                catch (...)
                {
                    std::cerr << "Error 11\n";
                    return 11;
                }
            }
        }
        catch (...)
        {
            std::cerr << "Error 6\n";
            return 6;
        }
    }
    catch (...)
    {
        std::cerr << "Error 5\n";
        return 5;
    }
}
else
{
    std::ifstream FileChck;
    try
```

```
        {
            FileChck.open(adr, std::ios_base::binary);
        }
        catch (...)
        {
            std::cerr << "Error 12\n";
            return 12;
        }
        try
        {
            FileChck.read(dva, 2);
        }
        catch (...)
        {
            std::cerr << "Error 13\n";
            FileChck.close();
            return 13;
        }
        if ((dva[0] == 77) && (dva[1] == 90))
        {
            ChIErr                                          =
check_file(std::filesystem::directory_entry(adr),
ByteOffset, Signat);
            if (ChIErr > 0)
            {
                std::cerr << "Error 14\n";
                FileChck.close();
                return 14;
            }
        }
```

```
        else
        {
            try
            {
                std::cout << "Это не формат Portable
Executable\n";
            }
            catch (...)
            {
                std::cerr << "Error 15\n";
                FileChck.close();
                return 15;
            }
        }
        try
        {
            FileChck.close();
        }
        catch (...)
        {
            std::cerr << "Error 16\n";
            return 16;
        }
    }
    return 0;
}

int check_file(std::filesystem::directory_entry AdrFile,
int& ByteOffset, char*& Signat)
{
```

```
int ChIErr = 0;
bool ChBErr = 0;
size_t ChStErr = 0;
bool itog = 0;
char* FileStr;
try
{
    ChBErr = AdrFile.exists();
    if (!ChBErr) throw 1;
}
catch (...)
{
    std::cerr << "Error 1\n";
    return 1;
}
if (ByteOffset < 0)
{
    std::cerr << "Error 2\n";
    return 2;
}
if (Signat==NULL)
{
    std::cerr << "Error 3\n";
    return 3;
}
size_t SigLen = strlen(Signat);
try
{
    FileStr = new char[SigLen + 1];
}
```

```
    catch (...)
    {
        std::cerr << "Error 4\n";
        return 4;
    }
    for (size_t i = 0; i < SigLen + 1; ++i) FileStr[i] =
0;
    std::ifstream CurrFile;
    try
    {
        CurrFile.open(AdrFile.path(),
std::ios_base::binary);
    }
    catch (...)
    {
        std::cerr << "Error 5\n";
        delete[] FileStr;
        return 5;
    }
    try
    {
        CurrFile.seekg(ByteOffset, std::ios_base::beg);
    }
    catch (...)
    {
        std::cerr << "Error 6\n";
        CurrFile.close();
        delete[] FileStr;
        return 6;
    }
```

```
    try
    {
        CurrFile.read(FileStr, SigLen);
    }
    catch (...)
    {
        std::cerr << "Error 7\n";
        CurrFile.close();
        delete[] FileStr;
        return 7;
    }
    try
    {
        CurrFile.close();
    }
    catch (...)
    {
        std::cerr << "Error 8\n";
        delete[] FileStr;
        return 8;
    }
    if (strcmp(Signat, FileStr) == 0)
    {
        try
        {
            std::cout << "СИГНАТУРА ОБНАРУЖЕНА В " <<
AdrFile.path() << '\n';
        }
        catch (...)
        {
```

```
                std::cerr << "Error 9\n";
                delete[] FileStr;
                return 9;
            }
        }
        delete[] FileStr;
        return 0;
}
```