

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. 0361

Бушуев Д.И.

Преподаватель

Халиуллин Р.А.

Санкт-Петербург

2021

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Бушуев Д.И.

Группа 0361

Тема работы: Приложение «Сигнатурный сканер»

Исходные данные: разработать на языке программирования С или С++ (по выбору студента) сигнатурный сканер. Сигнатурный сканер должен сканировать файлы и обнаруживать заданные образцы по наличию сигнатуры в файле. Минимальный размер сигнатуры должен 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, определение, примеры сигнатур, примитивные вирусы и антивирусы, реализация программы, основные сведения о программном обеспечении, реализованные функции, результаты тестирования программы, заключение, список использованных источников, приложение 1 — руководство пользователя, приложение 2 — блок-схема, приложение 3 — исходный код

программы.

Предполагаемый объем пояснительной записки:

Не менее 30 страниц.

Дата выдачи задания: 25.02.2021

Дата сдачи реферата: 07.06.2021

Дата защиты реферата: 11.06.2021

Студент

Бушуев Д.И.

Преподаватель

Халиуллин Р.А.

АННОТАЦИЯ

Задачей курсовой работы является создание приложения «Сигнатурный сканер» на языке программирования C. Приложение должно иметь консольный интерфейс, возможность ввода пути к файлам формата exe и txt. Приложение должно корректно обрабатывать все ошибки (ошибки ввода/вывода, выделения/освобождения памяти).

SUMMARY

The objective of the course work is to create a Signature Scanner application in the C programming language. The application must have a console interface, the ability to enter the path to exe and txt files. The application must correctly handle all errors (I / O errors, memory allocation / deallocation).

СОДЕРЖАНИЕ

	Введение	6
1.	Теоретическая часть	7
1.1.	Определение	7
1.2.	Примеры сигнатур	7
1.3.	Примитивные вирусы и антивирусы	7
2.	Реализация программы	9
2.1.	Основные сведения о программном обеспечении	9
2.2.	Реализованные функции	9
3.	Результаты тестирования программы	14
	Заключение	18
	Список использованных источников	19
	Приложение 1. Руководство пользователя	20
	Приложение 2. Блок-схема	23
	Приложение 3. Исходный код программы	24

ВВЕДЕНИЕ

Основной задачей курсовой работы является создание приложения для «Сигнатурный сканер» на языке программирования С. Это приложение нужно для реализации корректной обработки компьютером зараженный файл или нет.

Для успешного выполнения курсовой работы мне потребовалось воспользоваться источниками информации, найденными мной в сети Интернет. После написания приложения потребовалось протестировать работу и проверить ее с помощью одной из сред программирования. В результате выполнения курсовой работы были получены навыки работы с различными видами команд на языке С, а также методы оптимизированного ведения кода.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1. Определение

Сигнатура — целочисленная или текстовая константа, используемая для однозначной идентификации ресурса или данных. Такое число само по себе не несёт никакого смысла и может вызвать недоумение, встретившись в коде программы без соответствующего контекста или комментария, при этом попытка изменить его на другое, даже близкое по значению, может привести к абсолютно непредсказуемым последствиям. По этой причине подобные числа были иронично названы магическими. В настоящее время это название прочно закрепилось как термин.

1.2. Примеры сигнатур

Любой откомпилированный класс языка Java начинается с шестнадцатеричного «магического числа» 0xCAFEBABE. Второй широко известный пример — любой исполняемый файл ОС Microsoft Windows с расширением .exe начинается с последовательности байт 0x4D5A (что соответствует ASCII-символам MZ — инициалы Марка Зибовски, одного из создателей MS-DOS). Менее известным примером является неинициализированный указатель в Microsoft Visual C++ (начиная с 2005 версии Microsoft Visual Studio), который в режиме отладки имеет адрес 0xDEADBEEF.

В UNIX-подобных операционных системах тип файла обычно определяется по сигнатуре файла, вне зависимости от расширения его названия. Для интерпретации сигнатуры файла в них предусматривается стандартная утилита file.

1.3. Примитивные вирусы и антивирусы

В 1980-е годы начали появляться первые компьютерные вирусы. Большая часть вирусов заражала файлы типа COM, реже EXE. С появлением вирусов появились и антивирусные программы, которые были рассчитаны на примитивные вирусы того времени. Сигнатуры вирусов зачастую хранились в незашифрованном виде, это приводило к тому, что антивирус находил вирусы в

другом антивирусе. В то время примитивные антивирусы удовлетворяли требованиям того времени.

2. РЕАЛИЗАЦИЯ ПРОГРАММЫ

2.1. Основные сведения о программном обеспечении

Курсовая работа выполнена на языке программирования C. Использовалась операционная система Windows 10 и среда разработки Code::Blocks, компилятор — GNU g++.

2.2. Реализованные функции

1) Функция main

Функция main выводит на экран инструкцию по работе с программой, сообщения с просьбой ввести пути к файлам. После каждого сообщения вызывает функцию WTFcheck, задача которой проверить корректность введенного пути к файлу. Далее вызывается функция sign_read и, в зависимости от возвращаемого значения выводит сообщения о наличии/отсутствии последовательности байт в файле или сообщение об ошибке. Исходный код функции находится в файле code.c.

Объявление функции:

```
int main();
```

Тип функции: int.

Аргументы функции: функция аргументов не принимает.

Возвращаемое значение:

- 0 – Успешное завершение функции;
- 1 – Возникла ошибка функции printf;
- 2 – Возникла ошибка функции printf;
- 3 – Возникла ошибка функции WTFcheck;
- 4 – Возникла ошибка функции printf;
- 5 – Возникла ошибка функции printf;
- 6 – Возникла ошибка функции WTFcheck;
- 7 – Возникла ошибка функции printf;
- 8 – Возникла ошибка функции printf;
- 9 – Возникла ошибка функции sign_read;
- 10 – Возникла ошибка функции printf;

- 11 – Возникла ошибка функции printf.

Авторы функции: Бушуев Данил.

2) Функция WTFcheck()

В функции WTFcheck вводится путь к файлу, совершается его пробное открытие, определение формата файла (.exe или иной формат). Исходный код функции находится в файле code.c.

Объявление функции:

```
int WTFcheck(char* a, int* check);
```

Тип функции: int.

Аргументы функции:

- char* a – указатель на строку, в которую, в случае успешного завершения функции, записывается путь к файлу;
- int* check – переменная, через которую возвращаются значения в случае различных удачных исходах работы функции.

Возвращаемое значение:

- 0 – Успешное завершение функции;
- 1 – Возникла ошибка с передачей первого аргумента;
- 2 – Возникла ошибка с передачей второго аргумента;
- 3 – Возникла ошибка функции scanf;
- 4 – Возникла ошибка функции fflush;
- 5 – Возникла ошибка функции printf;
- 6 – Возникла ошибка функции fread;
- 7 – Возникла ошибка функции fclose;
- 8 – Возникла ошибка функции fclose.

Авторы функции: Иконников Степан.

3) Функция sign_read

Функция sign_read открывает текстовый файл со списком сигнатур, считывает из него первое значение – количество сигнатур в файле, затем в цикле for по количеству сигнатур считывает длину сигнатуры, ее смещение и саму

сигнатуру (последовательность байт), вызывает функцию `compare`. Исходный код функции находится в файле `code.c`.

Объявление функции:

```
int sign_read(char* A, char* E, int* checkSR);
```

Тип функции: `int`.

Аргументы функции:

- `char* A` – путь к файлу с сигнатурами;
- `char* E` – путь к файлу, в котором нужно искать последовательность;
- `int* checkSR` – переменная, через которую возвращаются различные значения функции `sign_read` при ее успешных завершениях.

Возвращаемое значение:

- 0 – Успешное завершение функции;
- 1 – Возникла ошибка с передачей 1 аргумента;
- 2 – Возникла ошибка с передачей 2 аргумента;
- 3 – Возникла ошибка с передачей 3 аргумента;
- 4 – Возникла ошибка с передачей 4 аргумента;
- 5 – Возникла ошибка функции `fopen`;
- 6 – Возникла ошибка функции `malloc`;
- 7 – Возникла ошибка функции `fscanf`;
- 8 – Возникла ошибка функции `fscanf`;
- 9 – Возникла ошибка функции `fscanf`;
- 10 – Возникла ошибка функции `compare`;
- 11 – Возникла ошибка функции `fclose`;
- 12 – Возникла ошибка функции `fclose`;
- 13 – Непредвиденная ошибка.

Авторы функции: Иконников Степан.

4) Функция `compare`

Функция `compare` открывает файл, в котором нужно найти сигнатуру, считывает его длину, проверяет, поместится ли сигнатура в файле. Если нет, то

переходит по смещению сигнатуры и сравнивает каждый байт сигнатуры, считанной из файла с сигнатурами с байтами в файле, который нужно проверить. Исходный код функции находится в файле code.c.

Объявление функции:

```
int compare(char* E, struct virus* sign, int* checkCMPR);
```

Тип функции: int.

Аргументы функции:

- char* E – путь к файлу, в котором необходимо найти сигнатуру;
- struct virus* sign – структура, содержащая смещение сигнатуры, ее длину и саму сигнатуру, считанную из текстового файла.

Возвращаемое значение:

- 0 – Успешное завершение функции;
- 1 – Возникла ошибка с передачей 1 аргумента;
- 2 – Возникла ошибка с передачей 2 аргумента;
- 3 – Возникла ошибка с передачей 3 аргумента;
- 4 – Возникла ошибка функции fopen;
- 5 – Возникла ошибка функции fseek;
- 6 – Возникла ошибка функции ftell;
- 7 – Возникла ошибка функции fclose;
- 8 – Возникла ошибка функции fseek;
- 9 – Возникла ошибка функции fread;
- 10 – Возникла ошибка функции fclose;
- 11 – Возникла ошибка функции fclose.

Авторы функции: Бушуев Данил.

5) Функция EmergencyFileClose

Функция EmergencyFileClose закрывает файл и проверяет его закрытие. Используется для экстренного закрытия файла в случае возникновения ошибки в предыдущих функциях. Исходный код функции находится в файле code.c.

Объявление функции:

`int EmergencyFileClose(FILE* f);`

Тип функции: `int`.

Аргументы функции:

- `FILE* f` – файл, который необходимо закрыть.

Возвращаемое значение:

- `0` – Успешное завершение функции;
- `1` – Возникла ошибка функции.

Авторы функции: Бушуев Данил.

3. РЕЗУЛЬТАТ ТЕСТИРОВАНИЯ ПРОГРАММЫ

При запуске программы, на экран выводится сообщение о том, что сперва нужно прочитать инструкцию, а затем по инструкции ввести путь к файлу, который необходимо проверить на наличие сигнатуры (рисунок 1).

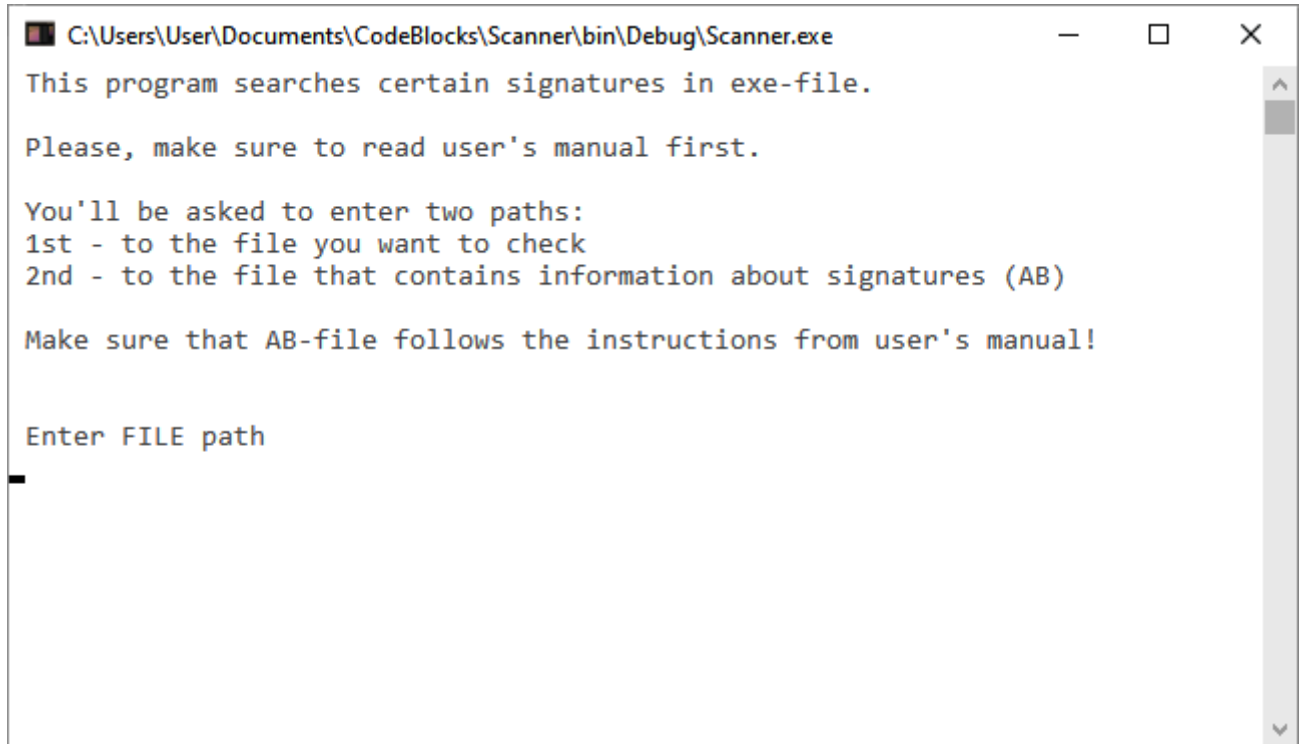


Рисунок 1 – Начало работы с программой

Далее требуется корректно ввести путь к файлу, в противном случае программа попросит ввести путь снова (рисунок 2).

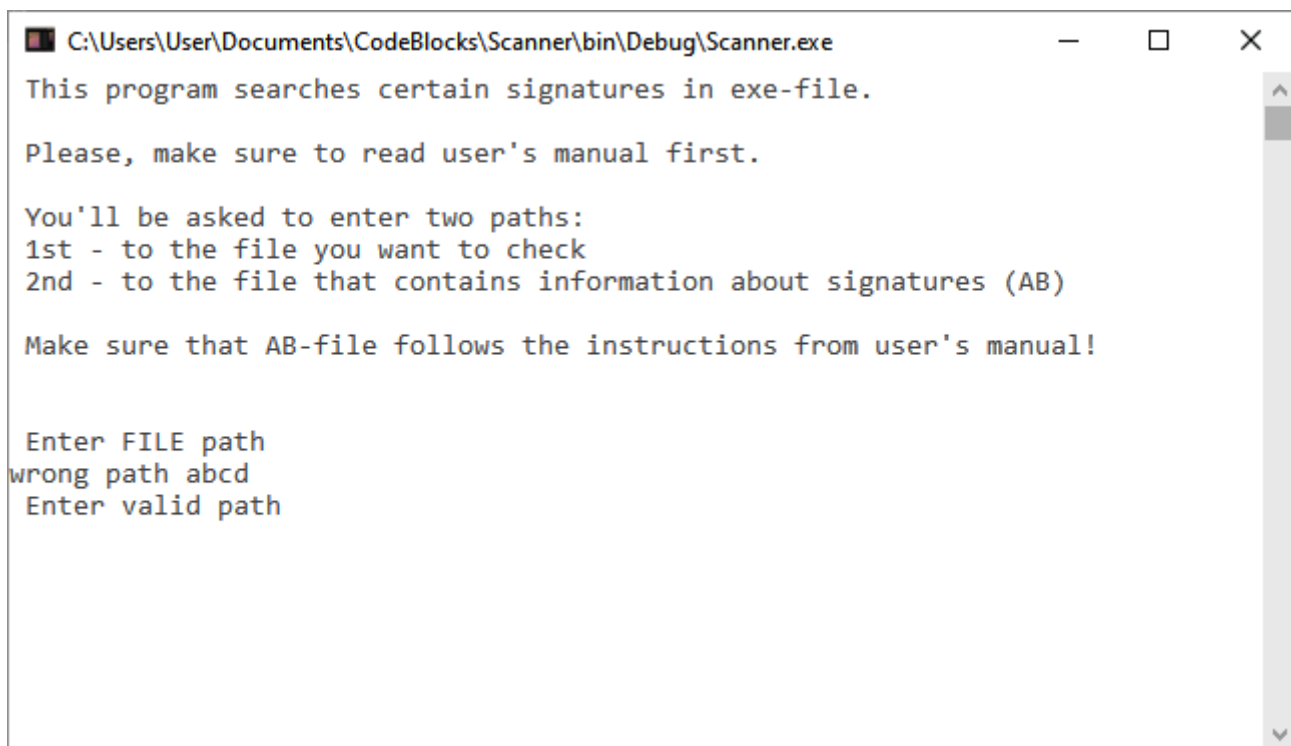


Рисунок 2 – Ошибочный ввод пути к файлу

После корректного ввода пути к файлу, программа попросит ввести путь уже к текстовому файлу, в котором записаны сигнатуры и их смещения (рисунок 3).

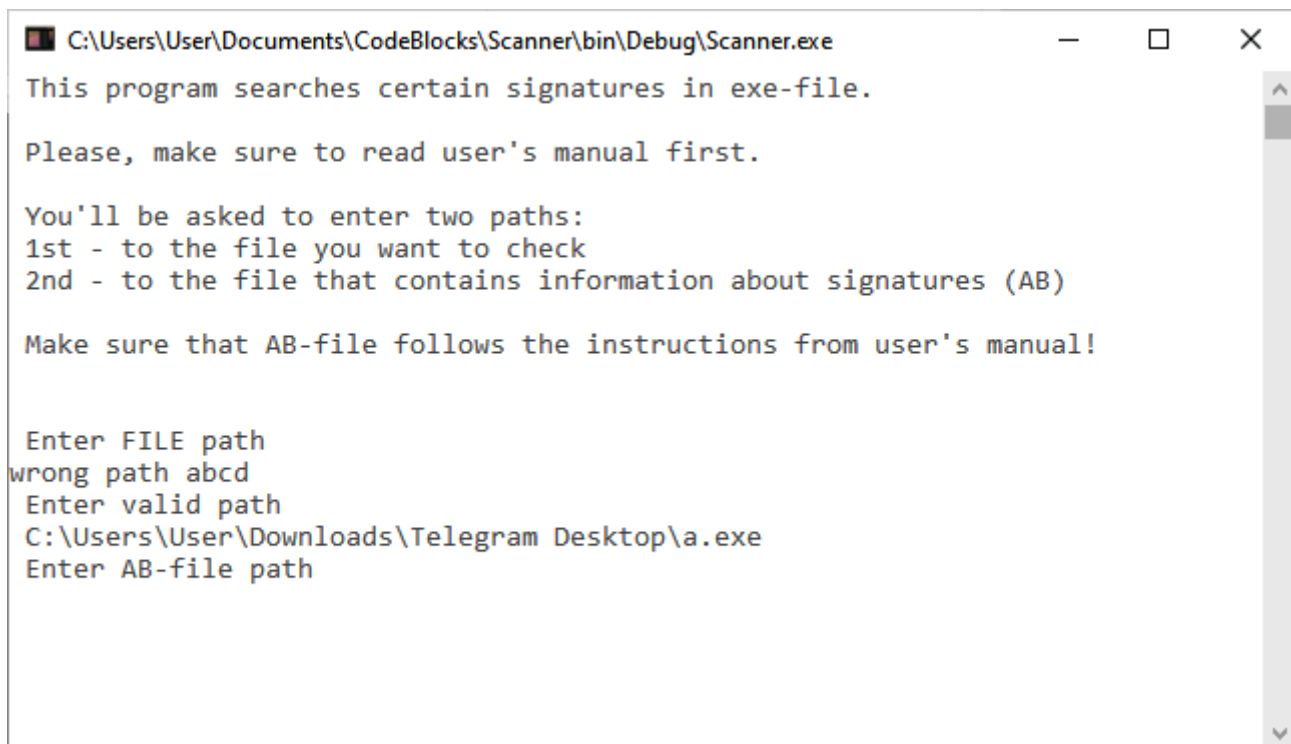
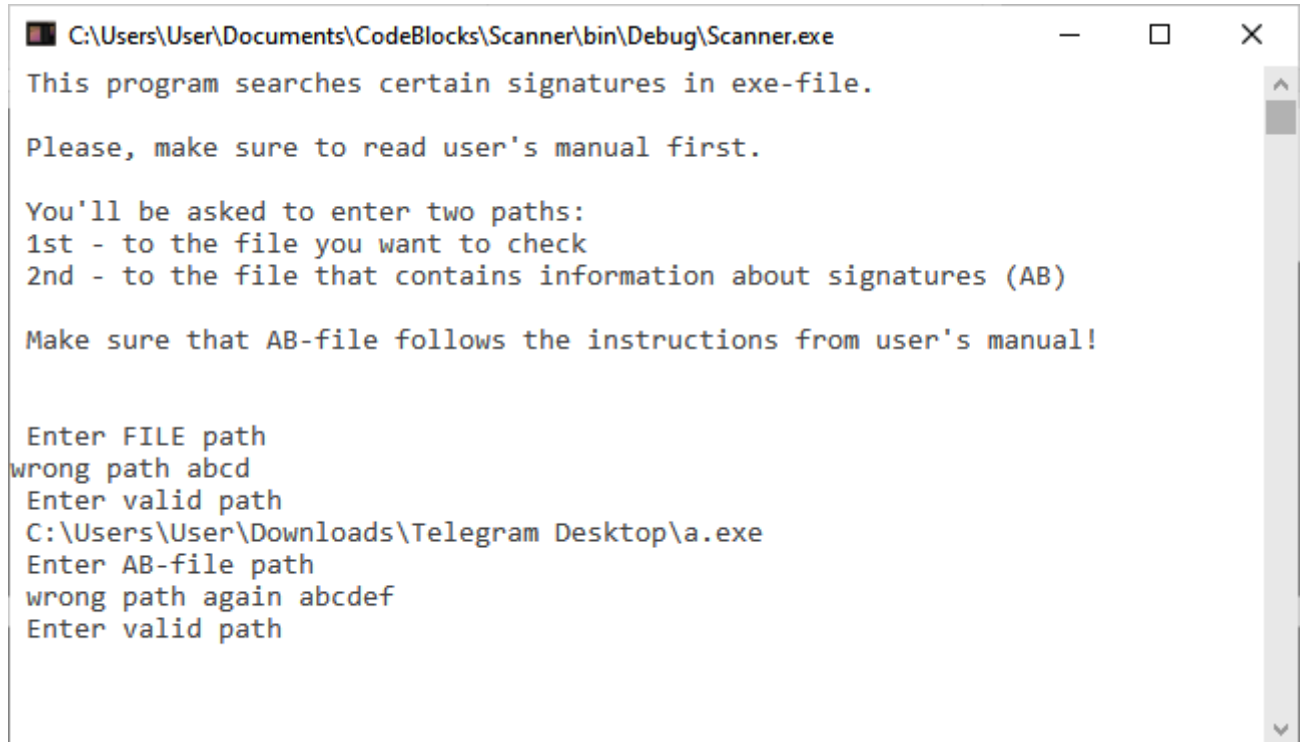


Рисунок 3 – Ввод пути к файлу с расширением .exe

Нужно ввести корректный путь к текстовому файлу, в противном случае программа попросит ввести путь снова (рисунок 4). После считывания пути к текстовому файлу программа выведет, что файл заражен (рисунок 5) или что файл безопасен (рисунок 6).



```
C:\Users\User\Documents\CodeBlocks\Scanner\bin\Debug\Scanner.exe
This program searches certain signatures in exe-file.

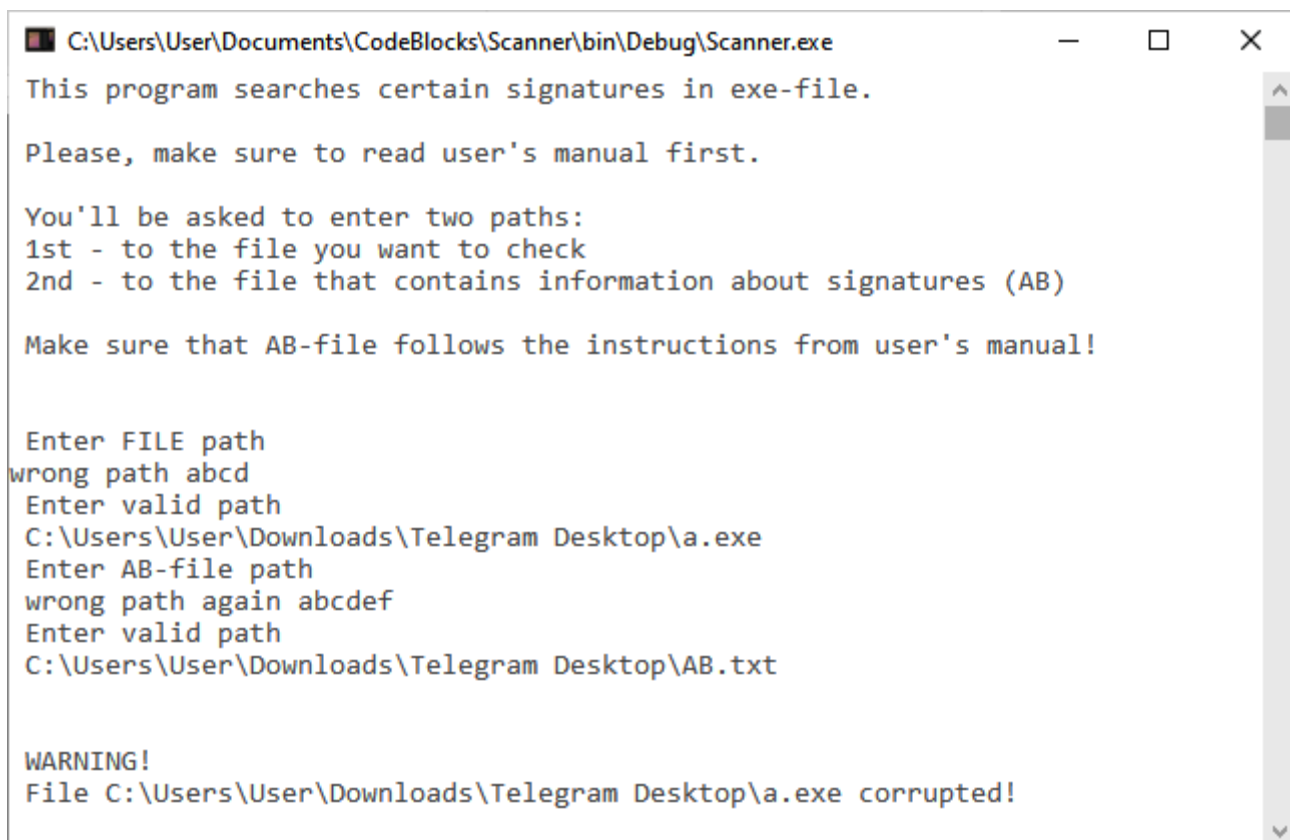
Please, make sure to read user's manual first.

You'll be asked to enter two paths:
1st - to the file you want to check
2nd - to the file that contains information about signatures (AB)

Make sure that AB-file follows the instructions from user's manual!

Enter FILE path
wrong path abcd
Enter valid path
C:\Users\User\Downloads\Telegram Desktop\a.exe
Enter AB-file path
wrong path again abcdef
Enter valid path
```

Рисунок 4 – Некорректный ввод пути к файлу с расширением .txt



```
C:\Users\User\Documents\CodeBlocks\Scanner\bin\Debug\Scanner.exe
This program searches certain signatures in exe-file.

Please, make sure to read user's manual first.

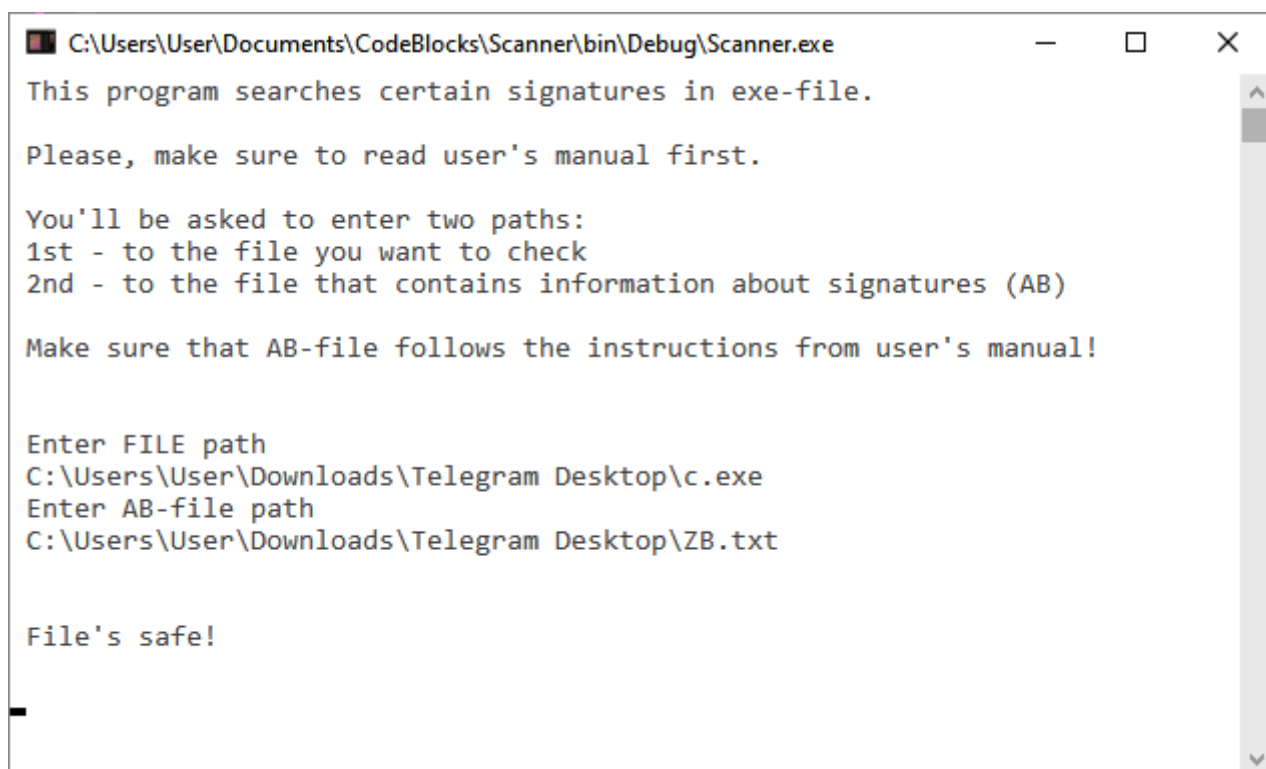
You'll be asked to enter two paths:
1st - to the file you want to check
2nd - to the file that contains information about signatures (AB)

Make sure that AB-file follows the instructions from user's manual!

Enter FILE path
wrong path abcd
Enter valid path
C:\Users\User\Downloads\Telegram Desktop\a.exe
Enter AB-file path
wrong path again abcdef
Enter valid path
C:\Users\User\Downloads\Telegram Desktop\AB.txt

WARNING!
File C:\Users\User\Downloads\Telegram Desktop\a.exe corrupted!
```

Рисунок 5 – Пример результата, что файл заражен



```
C:\Users\User\Documents\CodeBlocks\Scanner\bin\Debug\Scanner.exe
This program searches certain signatures in exe-file.

Please, make sure to read user's manual first.

You'll be asked to enter two paths:
1st - to the file you want to check
2nd - to the file that contains information about signatures (AB)

Make sure that AB-file follows the instructions from user's manual!

Enter FILE path
C:\Users\User\Downloads\Telegram Desktop\c.exe
Enter AB-file path
C:\Users\User\Downloads\Telegram Desktop\ZB.txt

File's safe!
```

Рисунок 6 – Пример результата, что файл безопасен

ЗАКЛЮЧЕНИЕ

В результате выполнения данной курсовой работы на языке программирования С было написано приложение «Сигнатурный сканер», которое имеет консольный интерфейс. Программа может считывать путь к файлу exe и путь к файлу txt, проверять корректность введенных данных, выводить на экран является ли файл зараженным или нет (есть ли в нем сигнатура из базы данных, находящейся в файле txt, или нет). Реализована корректная обработка ошибок, которые могут возникнуть в ходе выполнения программы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Керниган Б., Ритчи. Д. Язык программирования Си. — Санкт-Петербург: Невский диалект, 2000. — 352 с.
2. Standard C Library Functions Table, By Name [Электронный ресурс]. — URL: https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_74/rtref/stalib.htm (дата обращения: 05.06.2021).
3. The Open Group Technical Standard Base Specifications [Электронный ресурс]. — URL: <https://pubs.opengroup.org/onlinepubs/9699919799/> (дата обращения: 05.06.2021).
4. C/C++ Refences – All C Functions [Электронный ресурс]. — URL: https://doc.bccnsoft.com/docs/cppreference_en/all_c_functions.html (дата обращения: 05.06.2021).
5. Свободная энциклопедия [Электронный ресурс] — URL: [https://ru.wikipedia.org/wiki/Магическое_число_\(программирование\)](https://ru.wikipedia.org/wiki/Магическое_число_(программирование)) (дата обращения: 05.06.2021).
6. Свободная энциклопедия [Электронный ресурс] — URL: https://en.wikipedia.org/wiki/List_of_file_signatures (дата обращения: 05.06.2021).
7. Интернет издание cnews [Электронный ресурс] — URL: <https://www.cnews.ru/reviews/free/security2006/articles/evolution3/> (дата обращения: 06.06.2021).
8. Сайт антивирусного ПО [Электронный ресурс] — URL: <https://ru.malwarebytes.com/antivirus/> (дата обращения 06.06.2021).

ПРИЛОЖЕНИЕ 1. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Программа «Сигнатурный Сканер».

Программа предназначена для поиска определенной последовательности байт в исполняемом файле с расширением .exe.

Программа считывает с клавиатуры два пути: сначала путь к файлу exe, который нужно проверить, а затем путь к файлу txt, в котором хранятся сигнатуры, их длины и смещения. Перед началом программа выведет инструкцию на английском языке.

Файл, содержащий информацию о сигнатурах, должен быть записан следующим образом: 1 строка должна содержать десятичное положительное число N – количество сигнатур, записанных в файле. Последующие N строк должны содержать (строго соблюдая порядок) следующие положительные десятичные числа: смещение сигнатуры относительно начала файла, длину сигнатуры (K), K чисел, разделенных пробелом (байты сигнатуры).

Минимальные системные требования:

- Процессор - 1 ГГц и более с поддержкой PAE, NX и SSE2;
- Архитектура 32-бит или 64-бит;
- ОЗУ: 1 Гб (для 32-разрядных систем) или 2 Гб (для 64-разрядных систем);
- Видеоадаптер: DirectX версии не ниже 9 с драйвером WDDM 1.0;
- Дисплей: 800 x 600;
- Операционная система: Windows 10.

Установка программы:

Установка не требуется, нужно запустить исполняемый файл code.exe, скопировав его в любую директорию.

Запуск программы:

Запустить исполняемый файл code.exe

Работа с программой:

1. По инструкции, выведенной на экран, ввести путь к файлу с расширением .exe, который нужно проверить (рисунок 7).

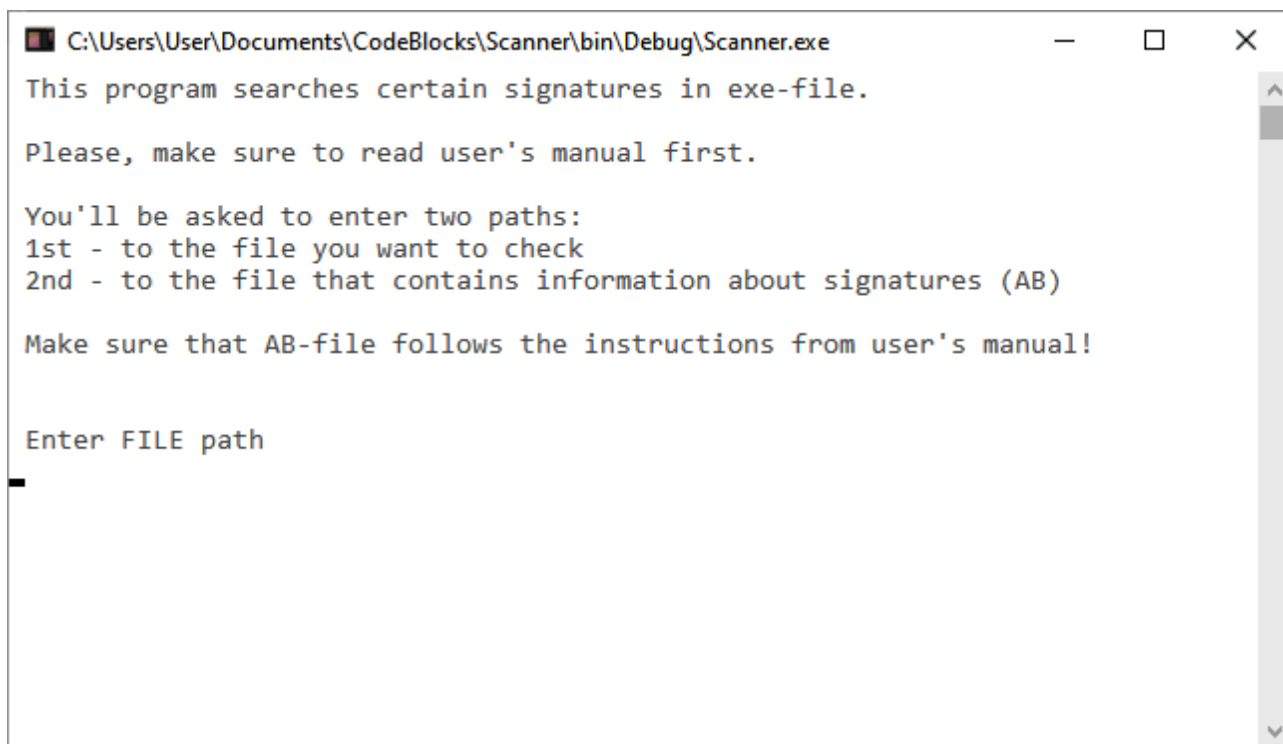


Рисунок 7 – Ввод пути к .exe файлу

2. По инструкции, выведенной на экран, ввести путь к файлу с расширением .txt, который нужно проверить (рисунок 8).

Путь к файлу не должен содержать знаки кириллицы.

Путь к файлу необходимо вводить вместе с расширением.

Длина пути не должна превышать 260 символов.

В ответ на Ваши действия программа выведет сообщение о наличии/отсутствии сигнатуры в файле или сообщение об ошибке.

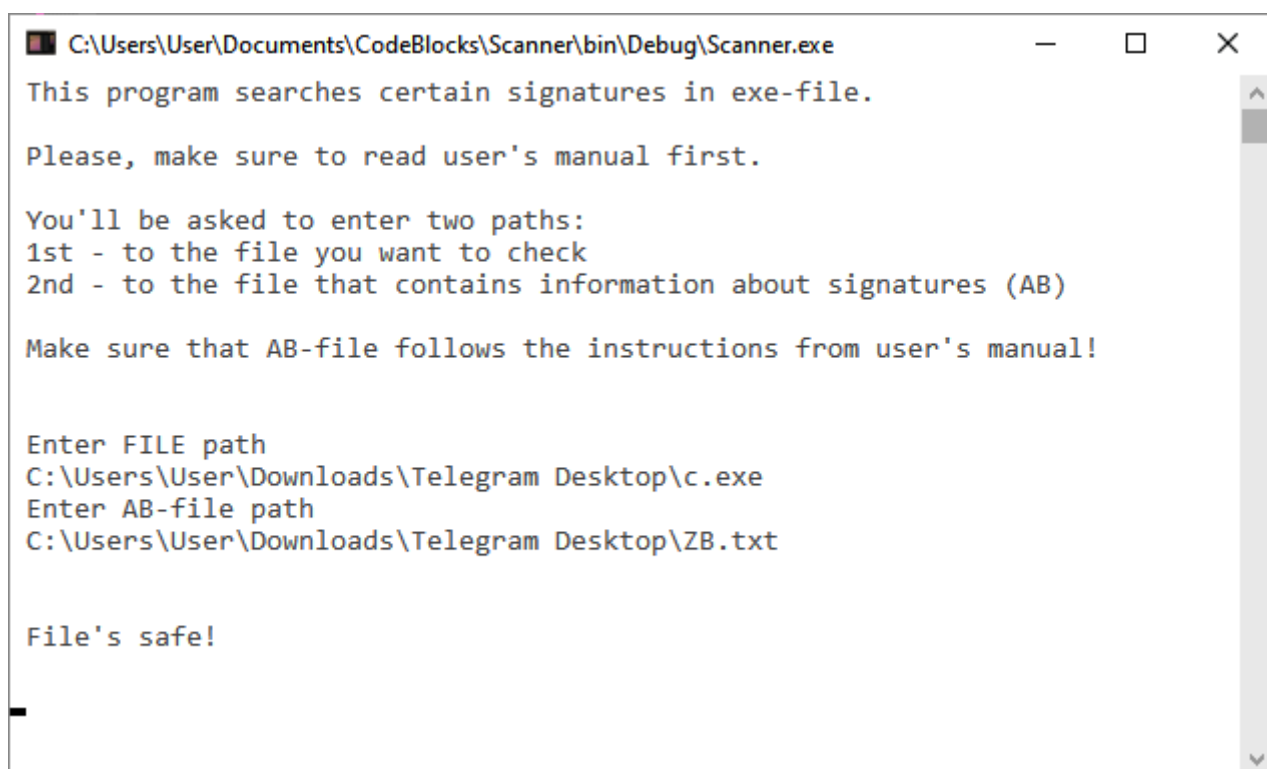
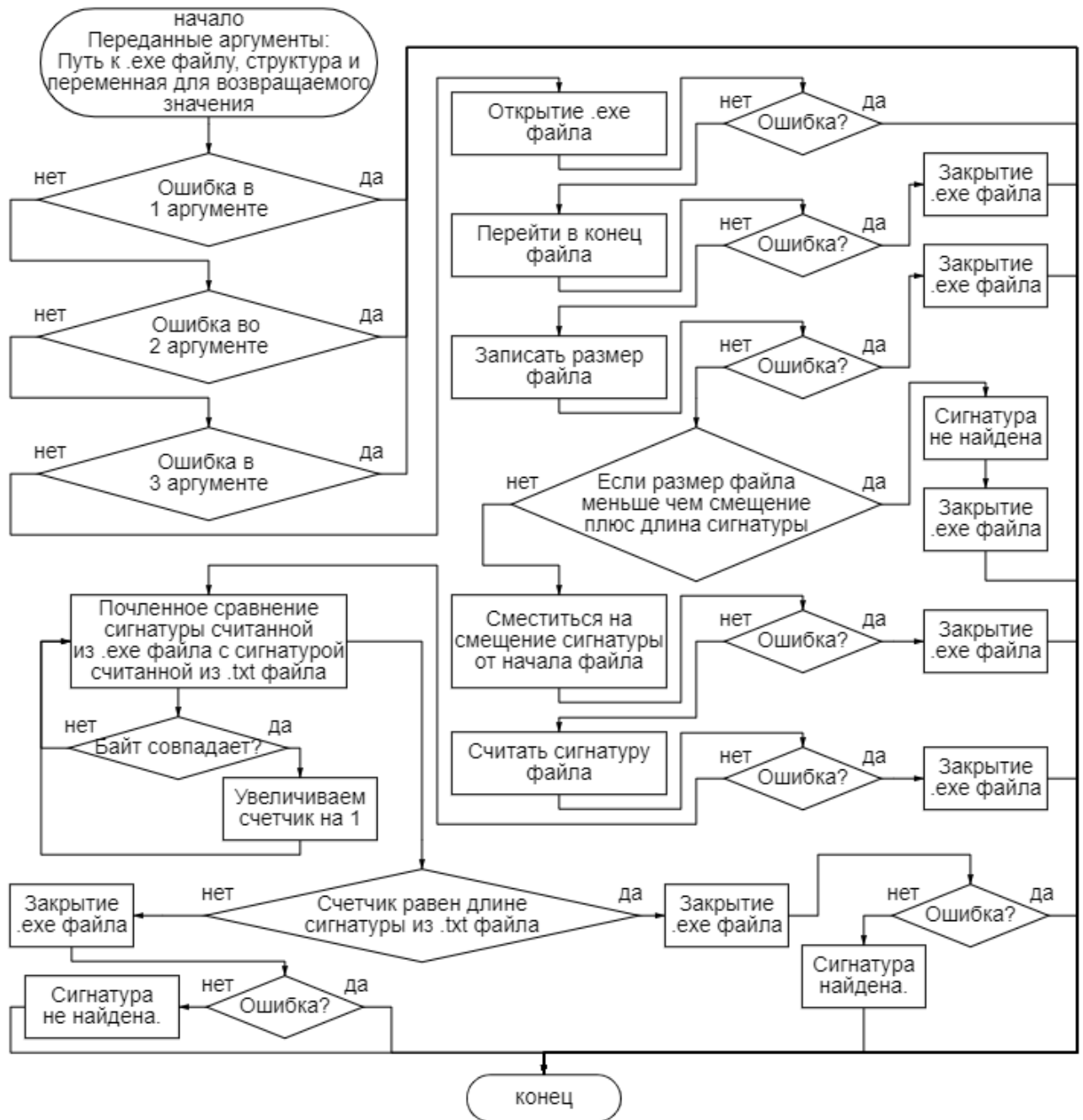


Рисунок 8 – Ввод пути к .txt файлу

ПРИЛОЖЕНИЕ 2. БЛОК СХЕМА



ПРИЛОЖЕНИЕ 3. ИСХОДНЫЙ КОД ПРОГРАММЫ

```
Файл code.c:
#define __USE_MINGW_ANSI_STDIO 1
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <conio.h>

int WTFcheck();
int EmergencyFileClose();
int compare();
int sign_read();

struct virus
{
    int length;
    int offset;
    unsigned char sign[1000];
};

int main()
{
    char pathExe[MAX_PATH];
    char pathTxt[MAX_PATH];
    int the_end, check, checkSR;

    if (printf(" This program searches certain signatures in exe-
file.\n\n Please, make sure to read user's manual first.\n\n
You'll be asked to enter two paths:\n 1st - to the file you want
to check\n 2nd - to the file that contains information about sig-
natures (AB)\n\n Make sure that AB-file follows the instructions
from user's manual!\n\n\n") < 0)
    {
        printf (" printf (26) error occurred (Is that even possi-
ble?!)\n");
        getch();
        return 1;
    }

    if (printf(" Enter FILE path\n ") < 0)
    {
        printf (" printf (33) error occurred (Is that even possi-
ble?!)\n");
        getch();
        return 2;
    }

    while (check != 13)
    {
        if (WTFcheck(pathExe, &check) != 0)
        {
```



```

        printf(" WayToFileCheck (42) error occurred\n");
        getch();
        return 3;
    }
    if (check != 13)
    {
        if (printf (" Enter valid path\n ") < 0)
        {
            printf (" printf (50) error occurred\n");
            return 4;
        }
    }

    if (printf(" Enter AB-file path\n ") < 0)
    {
        printf (" printf (58) error occurred (Is that even possible?!)\n");
        getch();
        return 5;
    }

    while (check != 31)
    {
        if (WTFcheck(pathTxt, &check) != 0)
        {
            printf(" WayToFile (67) error occurred\n");
            getch();
            return 6;
        }
        if (check != 31)
        {
            if (printf (" Enter valid path\n") < 0)
            {
                printf (" printf (75) error occurred\n");
                getch();
                return 7;
            }
        }
    }

    the_end = sign_read(pathTxt , pathExe, &checkSR);
    if (the_end != 0)
    {
        if (printf (" something went wrong, we'll fix that!\n") <
0)
        {
            printf (" printf (87) error occurred (Is that even possible?!)\n");
            getch();
            return 8;
        }
        getch();
    }

```

```

        return 9;
    }

    if (checkSR == 0 || checkSR == 1)
    {
        if (checkSR == 1)
        {
            if (printf ("\n\n File's safe!\n\n") < 0)
            {
                printf (" printf (101) error occurred (Is that
even possible?!)\n");
                getch();
                return 10;
            }
            getch();
            return 0;
        }
        if (checkSR == 0)
        {
            if (printf ("\n\n WARNING!\n File %s cor-
rupted!\n\n", pathExe) < 0)
            {
                printf (" printf (112) error occurred (Is that
even possible?!)\n");
                getch();
                return 11;
            }
            getch();
            return 0;
        }
    }
}

int WTFcheck(char* a, int* check)
{
    if (a == NULL)
    {
        printf(" WTFcheck argument 1 error occurred\n");
        return 1;
    }

    if (check == NULL)
    {
        printf(" WTFcheck argument 2 error occurred\n");
        return 2;
    }

    FILE* f;
    char path[MAX_PATH];
    char MN[2];
    int buffer, CKAH;
    size_t len;

```

```

while (1)
{
    CKAH = -1;
    len = 0;
    path[0] = '\0';
    CKAH = scanf("%[^\\n]s", &path);
    len = strlen(path);
    if (CKAH != 1 && len != 0)
    {
        printf(" scanf (149) error occurred\\n");
        return 3;
    }
    if ((fflush(stdin)) != 0)
    {
        printf (" fflush (156) error occurred\\n");
        return 4;
    }
    f = fopen((path), "rb");
    if (f != NULL) break;
    else if (printf(" Enter valid path\\n ") < 0)
    {
        printf(" printf (163) error occurred\\n");
        return 5;
    }
}

if (fread(MN, 1, 2, f) != 2)
{
    printf(" fread (170) ERROR occurred!\\n");
    EmergencyFileClose(f);
    return 6;
}
if (MN[0] == 'M' && MN[1] == 'Z')
{
    if ((fclose (f) != 0))
    {
        printf (" fclose (178) error occurred\\n");
        return 7;
    }
    for (size_t i = 0; i < len+1; i++)
    {
        a[i] = path[i];
    }
    buffer = 13;
    *check = buffer;
    return 0;
}
else
{
    if ((fclose (f) != 0))
    {
        printf (" fclose (193) error occurred\\n");
        return 8;
    }
}

```

```

    }
    for (size_t i = 0; i < len+1; i++)
    {
        a[i] = path[i];
    }
    buffer = 31;
    *check = buffer;
    return 0;
}
}

int sign_read(char* A, char* E, int* checkSR)
{
    if (A == NULL)
    {
        printf(" sign_read argument 1 error occurred\n");
        return 1;
    }

    if (E == NULL)
    {
        printf(" sign_read argument 2 error occurred\n");
        return 2;
    }

    if (checkSR == NULL)
    {
        printf(" sign_read argument 3 error occurred\n");
        return 3;
    }

    FILE* TXT;
    int check;
    struct virus* sign;
    int buffer;
    int N;
    int checkCMPR = -1;

    if ((TXT = fopen(A, "r")) == NULL)
    {
        printf (" fopen (235) error occurred\n");
        EmergencyFileClose (TXT);
        return 5;
    }

    if ((sign = (struct virus*)malloc(sizeof(struct virus))) ==
NULL)
    {
        printf(" malloc (242) error occurred\n");
        EmergencyFileClose (TXT);
        return 6;
    }
    if (fscanf(TXT, "%d", &N) != 1)

```

```

{
    printf(" fscanf (248) error occurred\n");
    EmergencyFileClose (TXT);
    return 7;
}
for (int i = 0; i < N; i++)
{
    if ((fscanf(TXT, "%d%d", &sign->offset,
&sign->length)) != 2)
    {
        printf(" fscanf (256) error occurred\n");
        EmergencyFileClose (TXT);
        return 8;
    }
    for (int j = 0; j < sign->length; j++)
    {
        if (fscanf(TXT, "%hhu", &sign->sign[j]) != 1)
        {
            printf (" fscanf (264) error occurred\n");
            EmergencyFileClose (TXT);
            return 9;
        }
    }
}

while(1)
{
    if (sign->length < 8) break;

    check = compare(E, sign, &checkCMPR);
    if (check != 0)
    {
        printf(" compare (276) error occurred\n");
        EmergencyFileClose (TXT);
        return 10;
    }
    if (checkCMPR == 0)
    {
        if ((fclose (TXT) != 0))
        {
            printf (" fclose (285) error occurred\n");
            return 11;
        }
        buffer = 0;
        *checkSR = buffer;
        return 0;
    }
    if (checkCMPR == 1) break;
}

}
if ((fclose (TXT) != 0))
{
    printf (" fclose (297) error occurred\n");
    return 12;
}

```

```

    }
    if (checkCMPR == 1)
    {
        buffer = 1;
        *checkSR = buffer;
        return 0;
    }
    return 13;
}

int compare(char* E, struct virus* sign, int* checkCMPR)
{
    if (E == NULL)
    {
        printf(" compare argument 1 error occurred\n");
        return 1;
    }

    if (sign == NULL)
    {
        printf(" compare argument 2 error occurred\n");
        return 2;
    }

    if (checkCMPR == NULL)
    {
        printf(" compare argument 3 error occurred\n");
        return 3;
    }

    FILE* EXE;
    int same = 0;
    int buffer;
    unsigned char exesign[1000];
    long int fileMaxOffset;

    if ((EXE = fopen(E, "rb")) == NULL)
    {
        printf (" fopen (337) error occurred\n");
        return 4;
    }

    if ((fseek(EXE, 0, SEEK_END)) == -1)
    {
        printf (" fseek (343) ERROR occurred\n");
        EmergencyFileClose (EXE);
        return 5;
    }

    if ((fileMaxOffset = ftell(EXE)) == -1)
    {
        printf (" ftell (349) error occurred\n");
        EmergencyFileClose (EXE);
        return 6;
    }

```

```

}

if (fileMaxOffset < (sign->offset + sign->length))
{
    buffer = 1;
    *checkCMPR = buffer;
    if (fclose(EXE) != 0)
    {
        printf (" fclose (360) error occurred\n");
        return 7;
    }
    return 0;
}

if ((fseek(EXE, sign->offset, SEEK_SET)) == -1)
{
    printf(" fseek (368) error occurred\n");
    EmergencyFileClose (EXE);
    return 8;
}
if (fread(exesign, 1, sign->length, EXE) != sign->length)
{
    printf (" fread (374) error occurred!\n");
    EmergencyFileClose (EXE);
    return 9;
}

for (size_t i = 0; i < sign->length; i++)
{
    if (exesign[i] == sign->sign[i]) { same++; }
}
if (same == sign->length)
{
    if ((fclose (EXE) != 0))
    {
        printf (" fclose (387) error occurred\n");
        return 10;
    }
    buffer = 0;
    *checkCMPR = buffer;
    return 0;
}
else
{
    if ((fclose (EXE) != 0))
    {
        printf (" fclose (398) error occurred\n");
        return 11;
    }
    buffer = 1;
    *checkCMPR = buffer;
    return 0;
}

```

```
}

int EmergencyFileClose(FILE* f)
{
    if (fclose(f) != 0)
    {
        printf(" fclose error occurred\n");
        return 1;
    }
    printf(" File was closed because some error occurred\n");
    return 0;
}
```