

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему РОЗРОБКА АНТИВІРУСНОЇ ПРОГРАМИ ДЛЯ ОПЕРАЦІЙНОЇ
СИСТЕМИ WINDOWS. ANTIVIRUS SOFTWARE DEVELOPMENT FOR
WINDOWS OS.

Виконав: студент(ка) 4 курсу, групи КНТ-227
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерні науки

Мирончук А.А.

(прізвище та ініціали)

Керівник Сердюк С.М.

(прізвище та ініціали)

Рецензент Неласа Г.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 122 Комп'ютерні науки
 (код і найменування)
 Освітня програма (спеціалізація) Комп'ютерні науки
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” _____ 2021 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТА(КИ)

Мирончука Артема Анатолійовича
 (прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розробка антивірусної програми для операційної системи Windows. Antivirus Software Development for Windows OS.
 керівник проекту (роботи) доц., к.т.н. Сердюк Сергій Микитович,
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
 затверджені наказом закладу вищої освіти від “30” березня 2021 року
 № 103
2. Строк подання студентом проекту (роботи) травня 2021 року
3. Вихідні дані до проекту (роботи) рекомендована література, технічне завдання
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань роботи. 2. Матеріали та методи. 3. Основні рішення щодо реалізації компонентів системи. 4. Керівництво програміста. 5. Керівництво оператора. Додатки.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Сердюк С.М., доцент		
Нормоконтроль	Дейнега Л.Ю., ст. викладач		

7. Дата видачі завдання 28 лютого 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Розробка архітектури програми.	4 тиждень	Розділ 2
4	Розробка програми.	5–6 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми.	7 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	8–9 тижні	Розділ 5, Додатки
7	Захист роботи.	10 тиждень	

Студент(ка)

(підпис)

Мирончук А.А.

(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Сердюк С.М.

(прізвище та ініціали)

(підпис)

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
115 с., 17 табл., 28 рис., 2 дод., 18 джерел.

C++, C#, WIN32 API, DLL, MVS, SQL SERVER, DLLIMPORT, ВІРУС,
СКАНУВАННЯ, КОНТРОЛЬНА СУМА, СИГНАТУРА, ДЕСКРИПТОР.

Об'єкт дослідження – процеси захисту інформації в операційній системі (ОС) Windows.

Предмет дослідження – методи захисту інформації від вірусів в ОС Windows.

Мета роботи – розробка програмного забезпечення, яке працює по запиті користувача для виявлення шкідливих програм, шляхом сигнатурного сканування.

Матеріали, методи та технічні засоби: база даних сигнатур вірусів; об'єктно-орієнтоване програмування; мови програмування C++ та C#; середовища розробки Microsoft Visual Studio 17, Microsoft Visual Studio 19 та Microsoft SQL Server; персональний комп'ютер (ПК) з процесором (x64) під управлінням ОС Windows 10.

Результати. Розроблено програмне забезпечення, що виконує антивірусну перевірку обраного користувачем файлу або на каталогу шляхом сигнатурного сканування. Також було реалізовано інструментарій для редагування антивірусної бази.

Галузь використання – захист інформації в ОС Windows.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 115 pages, 28 figures, 17 tables, 2 appendixes, 18 sources.

OPERATING SYSTEM, ANDROID, DEVICE, FILE, FILE SYSTEM, FILE MANAGER, ANDROID STUDIO, JAVA.

Object of study is a processes of data protection in the operating system (OS) Windows.

Subject of study are methods of data protection from viruses in OS Windows.

The purpose of the work is development of software that works at the user's request to detect malware by means of signature scanning.

Materials, methods and technical means: virus signature database; object-oriented programming; programming languages C++ and C#; development environments Microsoft Visual Studio 17, Microsoft Visual Studio 19 and Microsoft SQL Server; personal computer (PC) with a processor (x64) running Windows 10.

Results. An application has been developed that performs an anti-virus scan of a file selected by the user or on a directory by means of signature scanning. A toolkit for editing the anti-virus database was also implemented.

The field of use is the data protection in OS Windows.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ	9
1 Аналіз проблеми та постановка завдань роботи	11
1.1 Огляд типів та характеристик вірусу.....	11
1.2 Огляд існуючих антивірусних програм.....	13
1.3 Розробка технічного завдання.....	17
1.3.1 Підстава для розробки.....	17
1.3.2 Мета розробки та її призначення	17
1.3.3 Основні вимоги до програмного продукту	17
1.3.4 Вимоги до технічних засобів, операційної системи і мови програмування	17
1.3.5 Обмеження на установку і виконання програми	20
1.3.6 Вимоги до програмної документації	20
2 Матеріали та методи.....	21
2.1 Аналіз технічного завдання.....	21
2.2 Формування вимог до сканеру	23
2.3 Аналіз програмних засобів	21
2.3.1 Взаємодія з ОС Windows. Огляд особливостей Win API.....	21
2.3.2 Вибір мови програмування для створення сканеру та менеджера антивірусу	21
2.3.3 Огляд особливостей MS SQL Server.....	33
2.3.4 Огляд ієрархії класів Windows Forms.....	33
2.3.5 Огляд особливостей середовища розробки	35
2.4 Розробка архітектури системи	35
3 Основні рішення щодо реалізації компонентів системи	39
3.1 Розробка алгоритму роботи сканера.....	39
3.2 Основні рішення проєкту test.sln	41

3.3 Основні рішення проєкту AV_Manager.sln.....	46
3.4 Особливості розробки програми антивірусу	64
3.5 Розробка основних функцій сканування. Логіка пошуку вірусу.....	66
4 Керівництво програміста	72
4.1 Призначення і умови застосування розробленої програми	72
4.2 Характеристики програми	73
4.2.1 Бібліотеки та ідентифікатори, що використовувалися у проєкті test.sln	73
4.2.2 Підключення бібліотеки OpenSSL до проєкту test.sln.....	74
4.2.3 Бібліотеки та ідентифікатори, що використовувалися у проєкті AV_manager.sln	76
4.2.4 Функції Win32 API, що використовувалися у проєкті AV_manager.sln	76
4.2.5 Опис розроблених функцій бібліотеки CPPScanner.dll	78
4.3 Вхідні та вихідні дані	79
5 Керівництво оператора.....	81
5.1 Призначення програми.....	81
5.2 Умови використання програми	81
5.2.1 Вимоги до оператора.....	81
5.2.2 Вимоги до технічних засобів та операційної системи.....	81
5.3 Виконання програми	82
5.3.1 Запуск програми	82
5.3.2 Робота з антивірусною базою. Сканування.	84
5.4 Повідомлення оператору	86
Висновки.....	89
Перелік джерел посилання	90
Додаток А. Текст програми	92
Додаток Б. Копії слайдів презентації.....	106

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interfaces;
БД	– база даних;
ГГц	– гігагерц;
Кб	– кілобайт;
Мб	– мегабайт;
ОС	– операційна система;
ПК	– персональний комп'ютер;
ПЗ	– програмне забезпечення;
СКБД	– система керування базами даних;
ТЗ	– технічне завдання.

ВСТУП

Віруси з'явилися приблизно 50 років тому. Саме тоді, в кінці 60-х, в декількох «великих» комп'ютерах виявилися незвичайні програми, що не виконували розпорядження оператора та сильно сповільнювали роботу комп'ютера. З плином часу, з появою та широким розповсюдженням персональних комп'ютерів, а згодом і з появою мережі Internet, віруси набули силу та стали основною проблемою користувачів по всьому світі.

Завдання протидії наслідкам вірусів є предметом широких досліджень та дискусій це з кінця XX сторіччя. Тим паче у реаліях сучасного технічного прогресу ця тема додає своєї насущності та проблематики.

Отже, актуальність даної дипломної кваліфікаційної роботи бакалавра полягає у створенні програмного забезпечення, що має метою автоматизований захист файлів комп'ютера від дії шкідливих програм. І найбільш доцільним рішенням буде створення антивірусної програми.

Наразі, сучасний антивірус має в арсеналі велику кількість методів виявлення та протидії комп'ютерним вірусам, на вивчення та створення яких у експертів у сфері системного програмування вже витрачено десятки років. Серед таких методів найбільш увагу привертає метод сканування файлів за сигнатурами, через відносну простоту свого алгоритму та прозорість реалізації.

Сигнатура, в простому поданні, є послідовністю байтів, унікальною частиною в файлі. Ця частина повинна максимально однозначно виділяти файл серед множини інших файлів. Це означає, що обрана послідовність має бути присутня тільки одному файлу, якому вона належить, і цим відрізнити його від інших.

На практиці, крім самої послідовності, застосовуються ще додаткові параметри, що дозволяють максимально однозначно зіставляти сигнатуру з файлом. Введення додаткових параметрів також направлено на прискорення пошуку сигнатури в файлі. Такими параметрами, наприклад, можуть бути:

- розмір файлу;

- зсув послідовності байт;
- тип файлу;
- спеціальні маски (відображення того, як приблизно має виглядати файл, щоб в ньому могла міститися сигнатура, що шукається).

Отже, система що розробляється повинна виявляти небезпечні програми, серед усіх інших, методом сканування, а користувач, у свою чергу, піклуватися про обмеження виконання цих програм.

Практичне застосування системи має підвищити надійність зберігання інформації користувача на ПК.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ РОБОТИ

1.1 Огляд типів та характеристик вірусу

Комп'ютерний вірус – це спеціально написана, як правило, невелика за розмірами програма, яка може записувати свої копії в комп'ютерні програми, розташовані в виконуваних файлах, системних областях дисків, драйверах, документах і т. д., причому ці копії зберігають можливість до «розмноження».

Процес впровадження вірусом своєї копії в іншу програму називається зараженням, а програма або інший об'єкт, що містить вірус - зараженим.

Сьогодні науці відомо близько 30 тисяч комп'ютерних вірусів. Як і звичайним вірусам, комп'ютерним вірусам для «розмноження» потрібен «носіїв» - здорова програма або документ, в яких вони ховають ділянки свого програмного коду. Сам вірус невеликий, його розмір рідко вимірюється кілобайтами [1].

У той момент, коли користувач запускає на своєму комп'ютері заражену програму або відкриває документ, вірус активізується і змушує комп'ютер слідувати його, вірусу, інструкцій. Це призводить до видалення будь-якої інформації, причому найчастіше – безповоротно. Крім цього, сучасні віруси можуть зіпсувати не тільки програми, але і технічні засоби. Наприклад, знищують вміст BIOS материнської плати або ушкоджують жорсткий диск.

Комп'ютерні віруси бувають наступних типів [2]:

- файлові віруси, що вражають файли типу «exe» або «com». Першим заражається командний процесор, а через нього всі інші програми. Найбільш небезпечними є резидентні віруси, які залишаються в оперативній пам'яті постійно [3];

- завантажувальні або буткові віруси, що вражають завантажувальні сектори жорстких дисків та інших носіїв. Вони найбільш небезпечні для комп'ютера, так як в результаті їх руйнівної діяльності комп'ютер перестає завантажувати операційну систему;

- мережеві віруси, що вражають комп'ютери, які працюють в локальній або глобальній мережі;

- макровіруси, тобто програми, що представляють собою макроси, що використовуються в складних застосунках. В даний час макровіруси часто поширюються через різноманітні програми пакета Microsoft Office. Це відбувається тому, що файли документів і шаблонів включають в себе макроси. У якості прикладів макровірусів можна привести Npad, Cap, Wazzu;

- пакетні віруси є найбільш простим типом вірусів, у яких ядро програми міститься в командному пакетному файлі з розширенням bat.

Розглянемо способи захисту інформації від вірусів. Захист інформації – це комплекс заходів, спрямованих на забезпечення інформаційної безпеки [1], [4].

Антивірусна програма (антивірус) – комп'ютерна програма, яка призначена для знешкодження вірусів і різного роду шкідливого програмним забезпеченням (ПЗ), з метою збереження даних і забезпечення надійної роботи персонального комп'ютера.

Антивірусні програми (антивіруси) використовують два певних принципи роботи (усунення) з шкідливим ПЗ:

- сканування комп'ютера і зіставлення вже наявного вірусу з базою даних на сервері певного виробника;

- сканування і виявлення програм, які поведуться підозріло і можуть за визначенням бути шкідливим ПЗ.

Також можна визначити деяку класифікацію антивірусних модулів, які входять до складу різних антивірусних програм (антивірусів):

- сканери, тобто антивірусний модуль, який працює на основі зіставлення з відомими «описами» вірусів. Іншими словами, антивірус шукає наявність вірусу по базі сигнатур. Якість сканування залежить від дати поновлення баз даних і від евристичного аналізу;

- ревізорний модуль, який запам'ятовує стан файлової системи, що надалі дає можливість порівняти відмінності і зіставити результати. У випадку відхилення, знаходить вірус;

- монітори, тобто спеціальні програми помічники, які в разі виявлення потенційно небезпечного шкідливого ПЗ (найчастіше зустрічаються у EXE файлах) пропонують користувачеві на вибір декілька операцій, в число яких обов'язково входить функція "видалити";

- вакцини. Принцип дії модуля цього типу, може нагадувати звичайне "щеплення". Іншими словами, коли вірус хоче проникнути і заразити програму, то роль вакцини полягає в тому, щоб показати вірусу, що програма вже заражена (на даний момент даний спосіб застарів).

Основні завдання антивірусів:

- сканування файлів і програм в режимі реального часу;
- сканування комп'ютера на вимогу;
- сканування інтернет-трафіку;
- сканування електронної пошти;
- захист від атак ворожих веб вузлів;
- відновлення пошкоджених файлів (лікування).

1.2 Огляд існуючих антивірусних програм

В даному підрозділі проведено огляд наступних антивірусних програм:

- антивірусна програма лабораторії «Касперського» [4];
- «Avast Antivirus» [5];
- «AVG Antivirus» [6].

«Лабораторія Касперського», провідний розробник систем захисту від вірусів. Їх програмний продукт створений відповідно до концепції потрійного захисту, яка передбачає використання трьох основних методів виявлення загроз: сигнатурного, проактивного та евристичного [7].

Переваги програми:

- моніторинг та оновлення ПЗ;
- функція «Безпечне з'єднання» (безпечне підключення до громадських точок доступу);
- хмарні антивірусні технології (в основі технології лежить збір і аналіз даних від інших користувачів);
- система Watcher (знаходить шкідливі наслідки вірусів і усуває їх шляхом повернення системи в початковий стан);
- оновлений дизайн, виконаний в стилі Windows 10 [7].

Недоліки програми:

- невисока швидкість сканування (нижче ніж у деяких конкурентів);
- висока вартість (дорожче ніж інші конкурентні рішення).

Інтерфейс програми наведений на рис. 1.1.

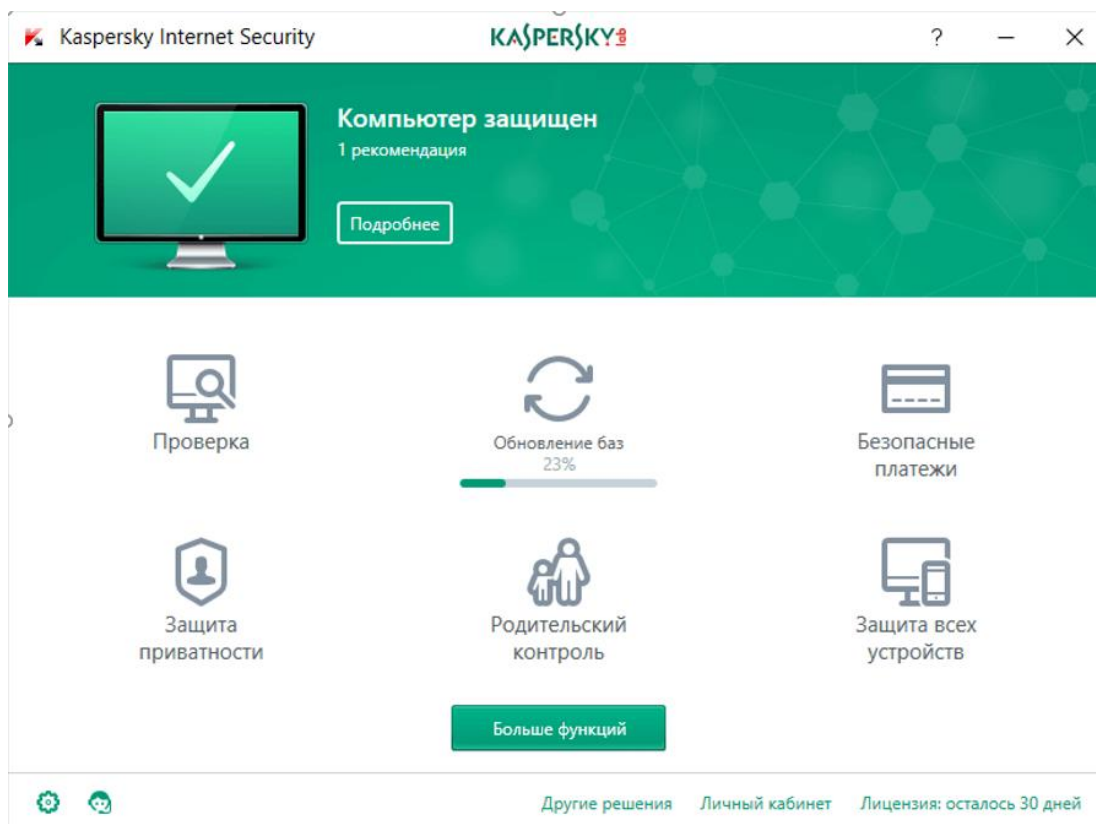


Рисунок 1.1 – Интерфейс программы «Kaspersky Internet Security 2017»

Avast Antivirus – це ціле сімейство антивірусних програм, розроблених компанією Avast для операційних систем Windows, Mac OS, Android і iOS. Воно включає в себе безкоштовну та комерційні версії продуктів, націлених на комп'ютерну безпеку та захист від шкідливих програм, а також інших типів кіберзагроз.

Переваги програм сімейства Avast Antivirus:

- сканер вірусів в реальному часі;
- автоматичне блокування підозрілих сайтів, а також фішингових ресурсів;
- аналіз на основі регулярно оновлюваних сигнатур, а також і евристичний аналіз;
- перевірки жорсткого диска безпосередньо через звертання до самих дисків, обходячи ОС Windows;
- оновлення відбуваються малими пакетами до 400 разів на добу;
- можливість створення диска для аварійного відновлення.

Недоліки програми:

- Avast в безкоштовній версії не можна використовувати для потреб бізнесу;
- має досить низькі за якістю сигнатурні поновлення, більше покладається на евристичні методи детектування загроз;
- при низькому рівні евристичного аналізу часто може помилково спрацьовувати.

Інтерфейс програми «Avast 2017» наведено на рис. 1.2.

Програма «AVG Antivirus» є антивірусною системою, виробництва чеської компанії AVG Technologies, що має сканер файлів, сканер електронної пошти і підтримує можливість автоматичного спостереження [8].

Переваги програми:

- захист від вірусів в реальному часі;
- Resident Shield (програма блокує шкідливе ПЗ);

- E-mail scanner (захищає листування по електронній пошті);
- безкоштовна;
- швидка перевірка всього комп'ютера;
- простий інтерфейс.

Приклад робота з програмою «AVG Free Antivirus» наведено на рис. 1.3.

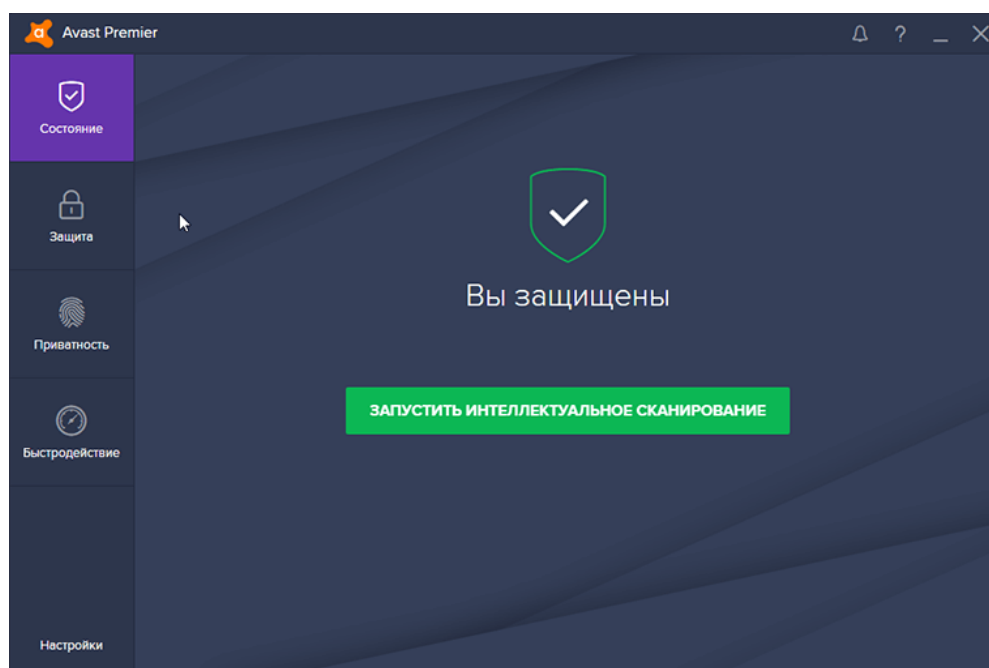


Рисунок 1.2 – Интерфейс программы «Avast 2017»

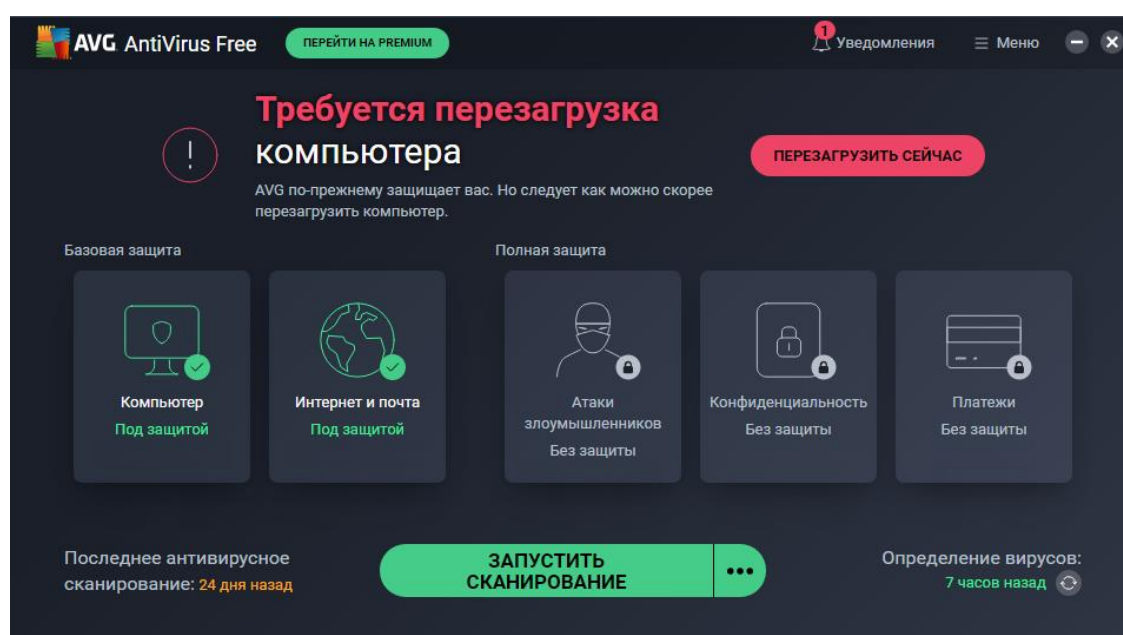


Рисунок 1.3 – Пример работы с программой «AVG Free Antivirus»

За результатами проведеного огляду програм аналогів антивірусів, необхідно розроблено технічне завдання на розробку власного застосунку.

1.3 Розробка технічного завдання

1.3.1 Підстава для розробки

Підставою для розробки служить завдання на дипломну кваліфікаційну роботу бакалавра на тему "Розробка антивірусної програми для операційної системи Windows", затверджене наказом № 103 від 30 березня 2021 р. по НУ «Запорізька політехніка».

1.3.2 Мета розробки та її призначення

Метою розробки є створення антивірусної програми, яка забезпечує виявлення шкідливих програм, що поширюються цільним файлом.

1.3.3 Основні вимоги до програмного продукту

Антивірусна програма, що розробляється повинна виявляти тільки шкідливі програми, що поширюються цільним файлом, тобто не заражають інші файли, як PE-віруси, і не змінюють своє тіло в процесі діяльності, як поліморфні віруси. Сканер повинен перевіряти файли шляхом порівняння сигнатур файлів з БД. Сигнатурою антивіруса є MD5-хеш (16 байт) згенерований на основі тіла відомого вірусу.

Антивірусна база має являти собою один або кілька файлів, що містять записи про всі відомі сканеру віруси. Вимоги до антивірусної бази мають визначитися її форматом, який в свою чергу залежить від інформації, що зберігається – записів антивірусної бази.

Кожен запис може містити:

- ім'я вірусу (користувач повинен знати ім'я вірусу);

- сигнатуру (обов'язково), по якій виконується пошук;
- додаткову інформацію, наприклад інструкції по лікуванню.

Сигнатура має складатися з:

- розміру послідовності ([Lenght * 4]). Для зберігання розміру послідовності відведемо 4 байти;
- хешу послідовності ([Hash * 16]). Для хешування будемо використовувати алгоритм MD5. Кожен MD5-хеш складається з 16 байт, або 4 подвійних слів.

Також сигнатура повинна містити зміщення послідовності в файлі ([Offset]).

Запис у антивірусній базі має містити:

- сигнатури ([Signature]);
- розмір імені файлу ([NameLen * 1]). Під розмір імені файлу виділимо 1 байт;
- ім'я файлу ([Name...]). Файл може бути довільного розміру до 255 символів включно.

Крім самих записів, в файлі бази повинен бути заголовок, в якому має міститися:

- число записів в базі ([RecordCount * 4]);
- сигнатура файлу антивірусної бази, щоб упевнитися, що це саме файл бази ([Sign * 3]).

Таким чином файл бази повинен мати структуру наведену на рис. 1.4.

Вхідними даними для програми мають бути ім'я обраного для перевірки файлу/каталогу та ім'я вірусу з БД.

Вихідні дані для програми сканеру:

- стан обраного для перевірки файлу/каталогу (інфікований чи ні);
- інформація про обраний для перевірки файл/каталог (шлях до файлу, розмір, стартову адресу у пам'яті);
- інформація про обраний вірус (ім'я, MD5-хеш, дескриптор);

- відредаговані записи про вірус у антивірусній базі.

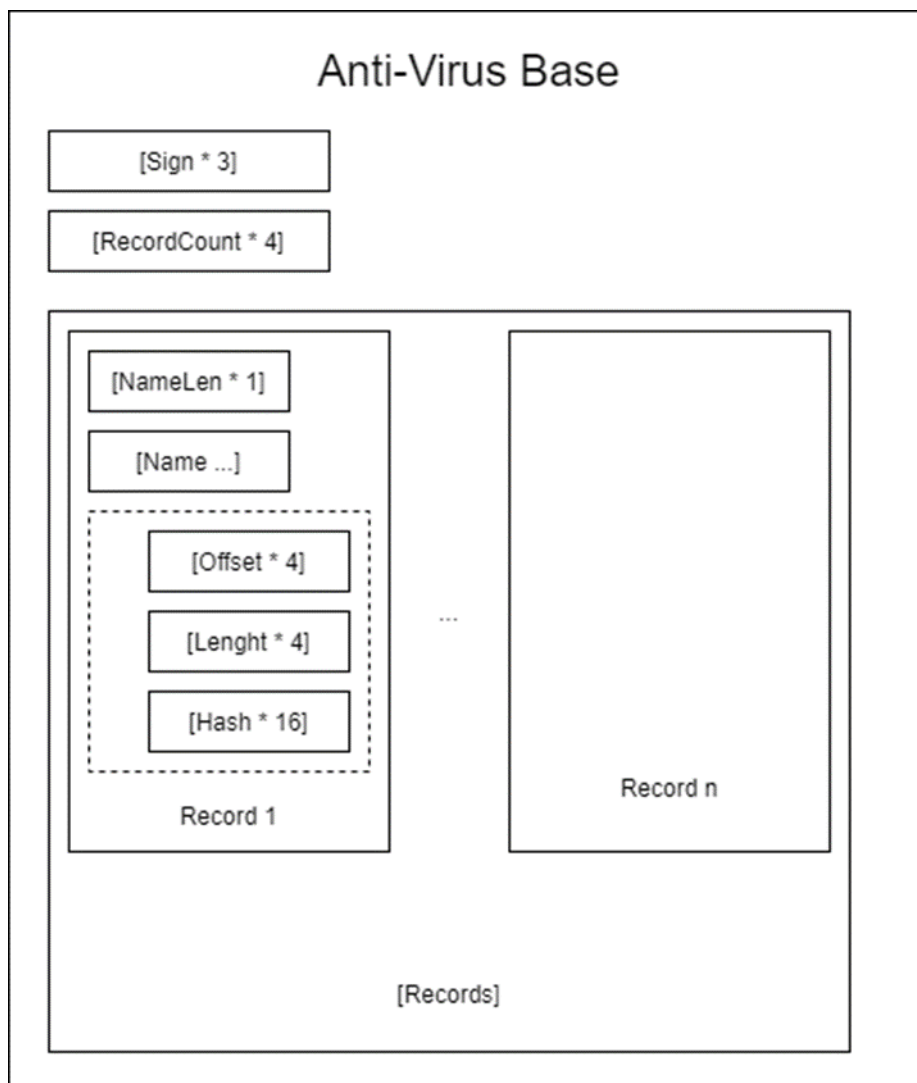


Рисунок 1.4 – Загальна структура файлу антивірусної бази

1.3.4 Вимоги до технічних засобів, операційної системи і мови програмування

Антивірус повинен бути розроблений під операційну систему Windows 10. Мінімальний набір апаратно-технічних засобів обумовлений мінімальними вимогами для забезпечення роботи ОС Windows 10:

- процесор 1 ГГц або швидший;
- оперативна пам'ять 1 Гб для 32-розрядних або 2 Гб для 64-розрядних ОС;

- місце на жорсткому диску 32 Гб для 64-бітної і 32-бітної ОС;
- відеокарта з підтримкою DirectX 9;
- розширення екрану 800x600, мінімальний розмір діагоналі для основного дисплея 7 дюймів або більше;
- маніпулятор типу миша.

Нижній рівень програми (ядро програми), тобто сам антивірус, необхідно написати з використанням мови програмування C++. Після написання програми на мові C++, користувач зможе взаємодіяти з нею за допомогою командного рядка. Для розробки ж користувацького інтерфейсу (верхнього рівня програми) необхідно використання мови C#, з підключенням сумісного проєкту на C++.

1.3.5 Обмеження на установку і виконання програми

Розроблена програма складається із модулів, кожен із яких сумісний з різними версіями ОС Windows (табл. 4.1). Тобто програма застосунку сумісна лише з останніми версіями Windows. Для того, щоб використати розроблений проєкт у попередніх версіях ОС, його потрібно перекомпілювати..

1.3.6 Вимоги до програмної документації

Програмна документація повинна включати: керівництво програміста, керівництво оператора та тексти основних програмних модулів.

2 МАТЕРІАЛИ ТА МЕТОДИ

2.1 Аналіз технічного завдання

Згідно з технічним завданням, програма антивірусу, що розробляється повинна виконувати головну функцію – сканування файлів комп'ютера. Цей процес нерозривно пов'язаний з поняттями:

- сигнатура, тобто послідовність байт, що є унікальною частиною в файлі. Ця частина повинна максимально однозначно виділяти файл серед множини інших файлів;
- контрольна сума, або хеш. Це отримана за допомогою алгоритму MD5, 16-байтова послідовність, що однозначно виділяє закодовану інформацію, серед всієї іншої;
- антивірусна база, тобто БД з сукупною інформацією про віруси.

Спрощений алгоритм роботи програми сканера, що відповідає вимогам ТЗ, наведений на рис. 2.1.

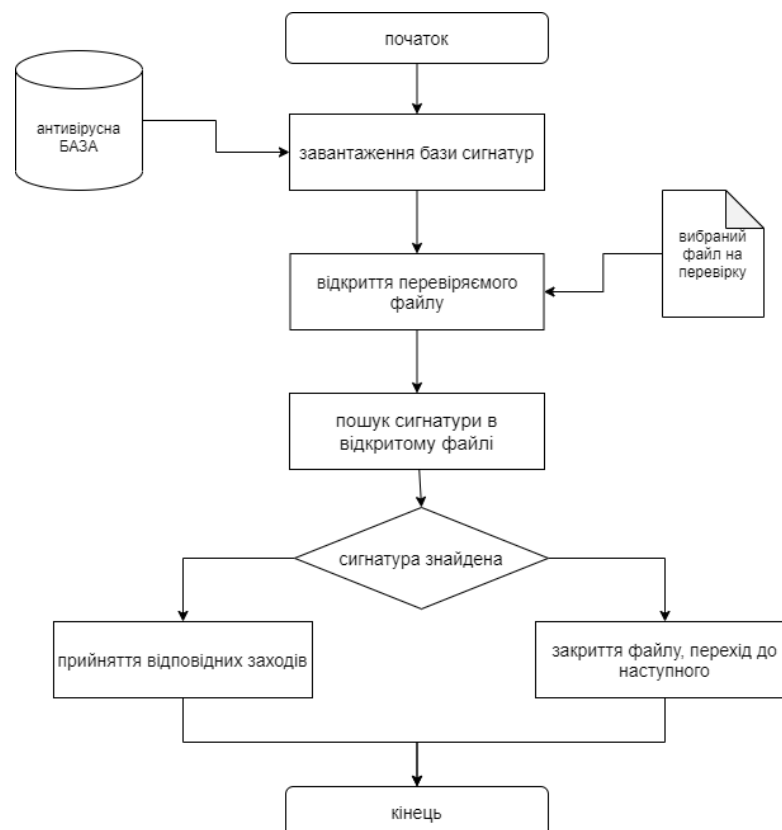


Рисунок 2.1 – Спрощений алгоритм роботи програми сканера

Структура алгоритму зумовлює модель майбутньої програми, що має складатися з трьох головних компонентів:

- антивірусна база даних;
- обраний для перевірки на безпечність файл;
- програма, що обробляє дані, та видає користувачу відповідь.

Більш того, ТЗ накладає вимоги на програмні засоби: антивірус повинен бути розроблений для ОС Windows 10, мовами програмування C++ та C#, з використанням можливостей Win32 API.

Створення повноцінної системи доцільно було б поділити на два взаємопов'язані проєкти. В такому разі, перший повинен забезпечувати усі базові функціональні можливості сканеру, тобто безпосередню перевірку файлу на наявність вірусу. Другий проєкт має виконувати роль «менеджеру», тобто з'єднувати користувача та «ядро» програми за допомогою інтерфейсу, а також забезпечувати ряд додаткових, зручних у використанні, функцій. Подібна теоретична модель зображена на рисунку 2.2.



Рисунок 2.2 – Двопроєктна теоретична модель системи

Таке розшарування системи на два проєкти пояснюється наступними мотивами:

- реалізація модульної розробки. У такому випадку умови створення структури проєктів будуть незалежними: для кожного проєкту різні залежності та структури даних, різні правила написання, а можливо і різні мови програмування. Це буде помітно полегшувати розробку та тестування системи. Модулі можна проєктувати не тільки в різних проєктах, але й різними людьми. Це може прискорити подальшу розробку, поліпшення та оптимізацію антивірусу в цілому;

- інкапсуляція. Фундаментальний принцип побудови програм в об'єктно-орієнтованому стилі. Інкапсуляція різних модулів, створених різними проєктами, полягає в розділенні відповідальності та доступності даних та методів. Таким чином, кожний проєкт буде відповідати лише за свої функції, які він може приховувати, або навпроти, робити відкритими для застосування;

- одночасна розробка. Якщо результатом розробки проєкту з менеджером буде вже готовий антивірус, то результатом розробки сканеру – бібліотека (більш точно, динамічного компонування). А файл з бібліотекою доволі легко підключити до верхнього проєкту, вказавши місце його розташування. Через таку легкість з'єднання, зміни у бібліотеки одразу будуть видимі у проєкті з менеджером. Це означає, що розробляти проєкти, вносячи зміни у код, можна паралельно, не витрачаючи часу на з'єднання частин.

2.2 Формування вимог до сканеру

Якщо вимоги до менеджера антивірусу визначаються поняттям зручності використання користувачем, то вимоги до сканеру можна сформулювати на базі технічного завдання. До таких вимог можна віднести:

- забезпечення доступу до використання інтерфейсу Win32. Сканер має бути розроблений з використанням функцій Win32 API, що у свою чергу мають бути доступними після підключення відповідних бібліотек;

- забезпечення приведення типів Win32. Програма обов'язково має бути створена з використанням типів даних Win32 API, питання лише у тому,

як ці типи даних привести до стандартних або користувацьких, в тій чи іншій мові програмування;

- забезпечення роботи під управлінням ОС Windows. Через те, що згідно з технічним завданням антивірус створюється для ОС Windows 10, сканер, як складова частина, теж має відповідати цій вимозі;

- забезпечення легкості створення бібліотек динамічного компонування. Під цим розуміють набір інструментів та засобів, що дозволяють створювати dll – результат опису поведінки сканеру;

- забезпечення додаткового функціоналу по роботі з файлами комп'ютера. Цілком задовольнити роботу з файлами Win32 не може – потрібні більш прості у застосуванні та звичні для середовища, методи роботи з файлами. Вони зручніші у розробці для відповідної мови програмування;

- забезпечення роботи з адресою. Через те, що в проєкті необхідно буде напряду працювати з адресою файлів, добре б було мати зручні засоби для роботи з нею;

- забезпечення роботи з пам'яттю. Керування пам'яттю знадобиться при скануванні великих масивів інформації, таких як директорії, або сканування не за одним вірусом, а зі всією базою;

- забезпечення роботи з проміжними даними. Для повної обробки результатів сканування, необхідно мати засоби для роботи з проміжними даними: для перегляду адрес файлів, їх вмісту, для перегляду вмісту 16-байтового хешу та для перевodu його у 32-символьний рядок;

- забезпечення використання кодування даних за алгоритмом MD5. Зокрема для отримання контрольної суми необхідно мати доступні інструменти для роботи з кодуванням інформації;

- забезпечення поєднання бібліотеки та оболонки. Після створення сканеру його функції необхідно в повній мірі перенести для використання у менеджері;

- швидка розробка, для виконання в зазначені строки;

- зручність відлагодження проєкту. Для перегляду результатів та для швидкого отримання рішення необхідні зручні інструменти відлагодження;
- кросплатформність. Сканер, що розробляється, як зазначено вище, має застосовуватись для Windows 10. Однак, його функціонал буде доречно застосовувати і на більш ранніх версіях ОС. Для цього лише потрібно змінити оболонку, або взагалі скористатися командним рядком.

2.3 Аналіз програмних засобів

В даному розділі проаналізовано вибір мови програмування, розглянуто основні особливості програмних засобів, за допомогою який реалізовано дипломну кваліфікаційну роботу.

2.3.1 Взаємодія з ОС Windows. Огляд особливостей Win API.

Windows API (англ. Application programming interfaces) – загальна назва набору базових функцій інтерфейсів програмування застосунків, операційних систем сімейств Microsoft Windows. Робота через Windows API - це найбільш близький до операційної системи спосіб її взаємодії з програмами [9].

Windows API представляє собою множину функцій, структур даних і числових констант. Всі мови програмування, здатні викликати такі функції і оперувати такими типами даних в програмах. Це мови: C ++; Pascal; Visual Basic та інші.

Win32 – 32-розрядний API для сучасних версій Windows. Найпопулярніша версія на даний момент. Win32 API оформлений у вигляді бібліотек динамічного компонування, що відрізняються від статичних бібліотек.

Статична бібліотека – це архів об'єктних файлів, який використовується в процесі статичної лінковки. Динамічна бібліотека це – виконуваний файл (тобто DLL і EXE в Windows мають однаковий формат).

Наступні файли складають більшу частину Win32 API: comdlg32.dll; dlllz32.dll; gdi32.dll; kernel32.dll; user32.dll та version.dll.

2.3.2 Вибір мови програмування для створення сканеру та менеджера антивірусу

Фактично вибір мови програмування для написання сканеру можна понятійно замінити вибором між кодом, що керується та некерованим. Додамо до цього твердження ще й той факт, що буде застосовано Win32 API, а оболонка буде створена засобами C#. Отримаємо повну задачу.

Для створення сканеру, стоїть вибір між кодом, що керується та некерованим у поєднанні з бібліотекою некерованого коду і оболонкою, написаною за кодом, що керується [10].

Кодом, що керується (managed), називають той код, що працює через загальномовне виконуюче середовище CLR, яке, крім іншого, пропонує такі служби, як збір сміття, перевірка типів під час виконання і перевірка посилань. Тобто такий код створюється засобами мови C#.

Некерований (unmanaged) код компілюється прямо в машинний код. Таким чином, за цим визначенням, весь код, скомпільований традиційними компіляторами C/C++ є некерованим.

Для порівняння мов C++ та C# було сформуємо критерії аналізу мов програмування у відповідності до вимог до сканеру. Результати порівняння мов C++ та C# за заданими критеріями наведено у таблиці 2.1.

Після аналізу за критеріями для мов C++ та C#, підсумуємо, що C++ забезпечує:

- більший рівень доступності до можливостей Win32 API, його функцій та типів даних;
- просте створення DLL;
- вимагає, в цілому, менше застосування технології «P/Invoke» та маршалінгу. Застосування при підключенні DLL;

Таблиця 2.1 – Порівняльний аналіз мов програмування для створення сканеру

Критерій оцінки	Мова програмування	
	C++	C#
1	2	3
Доступність функціоналу Win32 API	Функціонал Win32 API забезпечено напямую, через бібліотеку «windows.h». Ця бібліотека включена в стандартний пакет та не вимагає установки. В такому разі некерований код бібліотеки, взаємодіє з некерованим кодом проєкту.	Функціонал Win32 API забезпечено опосередковано, за допомогою технології «P/Invoke». Підключення здійснюється до бібліотеки kernel32.dll. Функція підключення DllImport доступна з простору імен System.Runtime.InteropServices та має бути застосована з коректними параметрами. В такому разі некерований код бібліотеки, взаємодіє з керованим кодом проєкту.
Створення DLL	У MVS 19 dll легко створюється або через проєкт відповідного типу, або через зміну типу конфігурації. Після створення проєкту виконується збірка. Докладніше про особливості створення див. у розділі 4.	У MVS dll легко створюється через проєкт відповідного типу. Після створення проєкту виконується збірка.

Продовження таблиці 2.1

1	2	3
Приведення типів Win32	Типи Win32 API також доступні з бібліотеки windows.h та мають свої прямі, стандартні аналоги у C++. MVS дозволяє одразу дізнатися про ці аналоги прямо у коді. Однак, окрім стандартних, в C++ можна безпосередню взаємодіяти і зі спеціальними типами, наприклад такими як дескриптори, або прапорами чи константами.	Типи Win32 API доступні також через інструмент «P/Invoke». Питання можуть виникати навіть при приведенні стандартних типів даних до типів Win32 API. Ці питання вирішує маршалінг даних [11]-[13]. Тип паче, у випадку спеціальних типів чи прапорів Win32 API, необхідно буде створювати користувацькі типи для приведення даних.
Робота під Windows 10	Робота під Windows 10 без проблем здійснюється за допомогою Microsoft Visual Studio. Детальніше – див. «кросплатформність».	Мова C# була створена працівниками компанії Microsoft. Тому під Windows утворилася досить повна .net інфраструктура.
Поєднання бібліотеки та оболонки	Якщо C# використовує «P/Invoke» для звернення до Win32 API, то C++ використовує цю технологію як раз для поєднання бібліотеки та оболонки.	Менеджер створений тією ж мовою, вимагає лише додання до проєкту DLL та застосування звичайного синтаксису мови для звернення до матеріалів бібліотеки.

Продовження таблиці 2.1

1	2	3
Додаткова робота з файлами	Робота з файлами реалізована через потоковий ввід-вивід. Однак додаткових, принципово спрощуючих створення проєкту, функцій для роботи з файлами, окрім звичайно бібліотеки windows.h, C++ не використовується.	Присутні широкі можливості у використанні функціоналу простору System.IO. Тут фактично є вся необхідна інформація про файли та, що дуже необхідна для опису та обслуговування роботи сканеру.
Робота з адресою	Фактично головна особливість мови C++ в даному контексті. Використання покажчика дозволяє посилатися не те місце в пам'яті, де зберігається файл, та отримувати його дані. А додаючи до нього зміщення в байтах, можна рухатися в адресному просторі. Тому, у подальшому будемо працювати з адресою файлу в просторі викликаного процесу, тобто самої програми.	Робота через покажчик засобами мовою C# можлива. Однак, при його використанні керований код одразу перейде в розряд некерованого. Так втратиться сенс використання C#, принаймні в цій роботі. Варто зауважити, що доцільно використовувати покажчик для оптимізації роботи програми, однак у меншій мірі, для основи сканеру.

Продовження таблиці 2.1

1	2	3
Робота з пам'яттю	Для сканеру, який оперує великими об'ємами адресного простору та пам'яттю, необхідне і чітке звільнення пам'яті. Такий функціонал C++ звісно що має, однак відповідальність за це накладається на програміста.	Передбачено автоматичний збір сміття. Для антивірусного сканеру такий підхід не є ефективним.
Робота з проміжними даними	Потоковий функціонал забезпечує форматований, «розумний» вивід для показчиків та типів Win32 API за допомогою перевантаження операторів. Це дуже вигідно застосовувати як для перегляду, так і для приведення, наприклад, показчика до рядка, засобами sstream.	Як і будь-яка робота C#, з даними некерowanego коду, тут викликає деякі «перенавантаження» на код, однак у кінцевому рахунку все ж таки забезпечує комфортний перегляд інформації, та приведення даних до потрібного типу (якщо ж це звісно можливо).
Можливості кодування даних	Кодування даних може здійснюватися за допомогою бібліотеки openssl/md5.h. Її функції дозволяють кодувати дані різними алгоритмами, серед яких є і MD5. Однак цю бібліотеку необхідно завантажити, скомпілювати та встановити. Цей процес описано у розділі 5.	Вбудований модуль System.Security.Cryptography у надає можливість кодування даних, необхідну для сканеру. Однак його доступність забезпечена за рахунок швидкості виконання, тим паче при великих обсягах даних у контексті сканеру.

Продовження таблиці 2.1

1	2	3
Швидкість розробки	Коли інфраструктура проєкту створена, основні підходи та бібліотеки обрані, швидкість розробки на C++ та швидкість розробки на C# стають приблизно однаковими.	C# дозволяє стартувати розробку швидше, а це дозволяє швидше отримати прототип рішення.
Зручність відлагодження	Звичайне відлагодження некерованого коду C++.	Звичайно, загалом відлагодження засобами C# зручніше. Однак, при розробці сканеру маємо використання некерованого коду (Win32 API), а звідси і більш незручне та складніше відлагодження.
Крос-платформність	C++ фактично кросплатформна, хоча і з деякими, додатковими витратами, а також бінарною несумісністю між платформами.	C#, реально, виявився не кросплатформним, незважаючи на існування неофіційних .net оточень під різними платформами і навіть потенційну бінарну сумісність між платформами.

- легко працює під Windows та фактично є кросплатформним;
- забезпечує роботу з адресою та безпосередню взаємодію з пам'яттю;

- може забезпечити додатковий функціонал для роботи з файлами та аналізу проміжних даних;

- надає швидкі алгоритми кодування інформації;

- зручний при відлагодженні;

- швидкий для розробки проєкту в цілому.

Для C# характерне:

- тотальне застосування технології «P/Invoke» та маршалінгу для Win32 API, що веде до нагромадження коду та більших складностей у використанні;

- складна схема роботи з типами даними некерованого коду, при їх великих об'ємах;

- легко працює під Windows 10;

- легко дозволяє створити та підключити бібліотеки динамічного програмування;

- забезпечує складніші та глибші додаткові можливості роботи з файлами;

- не має інструментів для безпечного керування пам'яттю та як наслідок, роботи з адресами файлів;

- повільніша робота алгоритмів кодування;

- зручний при відлагодженні.

Таким чином, для написання сканеру, а точніше його бібліотеки, доцільніше використати некерований код, тобто мову C++.

Особливостями мови C#, що у підсумку призвели до використання її для створення програми менеджера антивірусу є:

- приналежність до сім'ї мов з C-подібним синтаксисом, що найбільш близькі до C++ [14];

- наявність атрибутів DllImport і DllExport для взаємодії з файлами DLL [15];

- автоматичний контроль пам'яті, що стає актуальний на більш вищому рівні програмування проєкту;

- бібліотеки, що надають широкі можливості маніпулювання файлами;
- відносна легкість створення бібліотек;
- розробка користувацького інтерфейсу за допомогою програмного інтерфейсу Windows Forms.

2.3.3 Огляд особливостей MS SQL Server

SQL Server є однією з найбільш популярних систем керування базами даних (СКБД) в світі. SQL Server, як і множина інших програмних засобів, використаних у даній роботі, був створений компанією Microsoft.

SQL Server довгий час був винятково системою управління базами даних для Windows, проте починаючи з версії 16, ця система доступна і на Linux. SQL Server необхідно використати в даній роботі виходячи з наступних її особливостей:

- продуктивність. SQL Server працює дуже швидко;
- безпека. SQL Server надає шифрування даних;
- простота. З даною СКБД відносно легко працювати і вести адміністрування;
- доступність. Середовище розробки Visual Studio надає можливість створення, підключення та редагування БД, написаною за допомогою SQL Server, прямо у своєму середовищі. До того ж на мові C# є необхідні пакети бібліотек для взаємодії з цією СКБД.

2.3.4 Огляд ієрархії класів Windows Forms

Windows Forms – інтерфейс програмування застосунків (API), що відповідає за графічний інтерфейс користувача і є частиною Microsoft .NET Framework. Даний інтерфейс спрощує доступ до елементів інтерфейсу Microsoft Windows за рахунок створення обгортки для існуючого Win32 API.

Розробка інтерфейсу користувача забезпечується функціями класів, які створюють наступну ієрархію, наведену на рисунку 2.3.

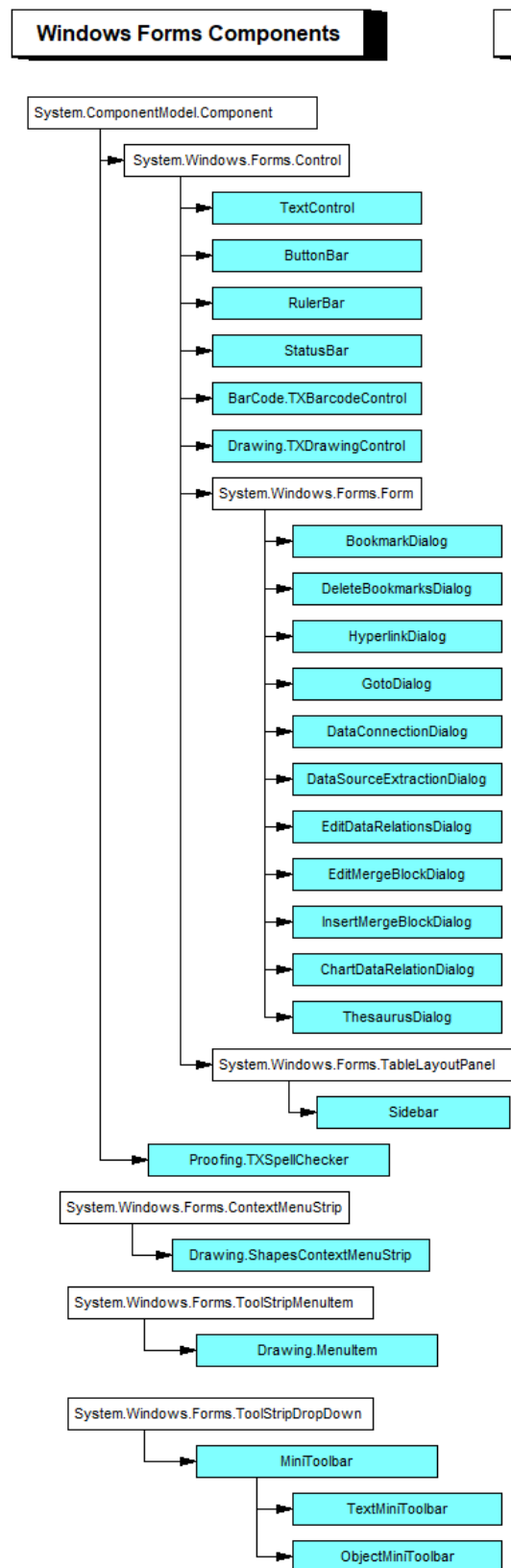


Рисунок 2.3 – Ієрархія класів Windows Forms

2.3.5 Огляд особливостей середовища розробки

Microsoft Visual Studio - лінійка продуктів компанії Microsoft, що включають інтегроване середовище розробки програмного забезпечення і ряд інших інструментальних засобів. Дані продукти дозволяють розробляти як консольні застосунки, так і застосунки з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, WPF, а також веб-сайти.

Visual Studio включає в себе редактор вихідного коду з підтримкою технології IntelliSense (технологія автодоповнення) і можливістю найпростішого рефакторінга коду.

Visual Studio дозволяє створювати і підключати сторонні програми (плагіни) для розширення функціональності практично на кожному рівні, включаючи додавання підтримки систем контролю версій вихідного коду.

2.4 Розробка архітектури системи

Перед проєктуванням архітектури системи доцільно було б виділити вимоги користувача до антивірусної програми. Для цього було розроблено діаграму прецедентів програми менеджера антивірусу наведену на рис. 2.4. Ця модель формально зображає різноманітні сценарії взаємодії між акторами (користувачами) і прецедентами (випадками використання), а також описує основні функції, які мають бути реалізовані в системі та вбудовані в програму менеджера антивірусу.

Архітектура системи, що розробляється зображена на рисунку 2.5.

Розробка антивірусної програми проводиться шляхом розробки двох проєктів:

- test.sln (на рис. 2.5 позначений блакитним кольором) – має бути написаний на мові C++, проєкт буде використовувати бібліотеку Win32 API та забезпечувати створення CPPScanner.dll з необхідним функціоналом для сканування файлу;

– AV_Manager.sln (на рис. 2.5 позначений червоним кольором) – має бути написаний на мові C#. Проєкт буде використовувати функціонал CPPScanner.dll та забезпечувати створення безпосередньо антивірусу, з графічним інтерфейсом та всіма необхідними можливостями для проведення та аналізу результатів сканування.

Як видно з рис. 2.5, архітектуру антивірусної системи спроектовано за багаторівневим шаблоном.



Рисунок 2.4 – Діаграма прецедентів менеджера антивірусу

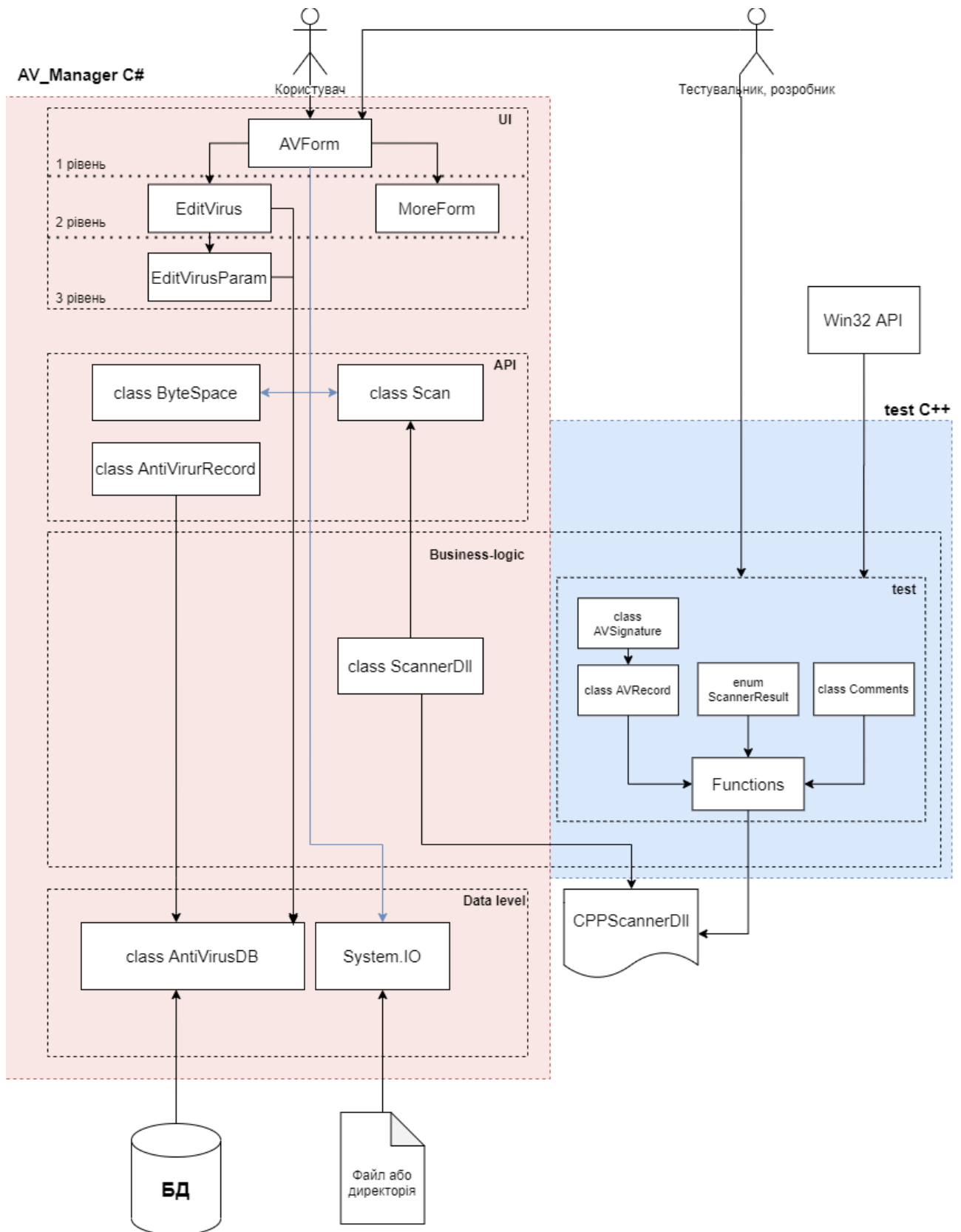


Рисунок 2.5 – Архітектура системи

В цьому випадку маємо наступні чотири рівні:

– рівень подання, або користувацький інтерфейс («UI» на рис. 2.5) охоплює все, що має відношення до спілкування користувача з системою. Він складається з головної форми, та трьох їй підпорядкованих форм. Тут варто відмітити, що для розробника рівень подання надає основу для відлагодження та тестування ПЗ. Додатково, розробник має використовувати командний рядок, для отримання більш повної інформації;

– рівень застосунку, або рівень сервісу, операційний рівень, рівень служб («API» на рис. 2.5). Логіка рівня подання взаємодіє з бізнес-логікою виключно за посередництво рівня служб, який діє як API застосунку. В даному випадку, рівню служб передано програмні інтерфейси, що забезпечують доступ до сканування, взаємодію з байтовим представленням та поняттям вірусу у системі;

– рівень бізнес-логіки, або рівень предметної області («Business logic» на рис. 2.5) описує основні функції програми, призначені для досягнення поставленої перед нею мети. До таких функцій відносяться обчислення на основі введених і збережених даних: тобто тут знаходиться функціонал, що повністю забезпечує сканування файлу на наявність вірусу;

– рівень доступу до даних, або рівень зберігання даних («Data level» на рис. 2.5) забезпечує доступ та встановлення зв'язку з БД, експорт та імпорт даних. Тобто тут відбувається взаємодія системи з антивірусною базою та файлом, що перевіряється або сукупністю файлів.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Розробка алгоритму роботи програми сканера

Умовно алгоритм сканеру можна поділити на дві функціональні частини:

- пошук, в адресному просторі процесу, початку файлу, який необхідно перевірити;

- визначення за знайденим покажчиком контрольної суми необхідних байтів файлу, та перевірка її з хешем вірусу. Повернення результату.

Алгоритм роботи програми сканеру наведено на рис. 2.1 та схемі 2.2.

Блок-схема алгоритму програми, що забезпечує виконання передбачених технічним завданням функцій наведена на рис. 3.1 та рис. 3.2.

Почнемо з розгляду першої частини. У ній відбувається пошук початку файлу в адресному просторі процесу, що був викликаний. Тобто відбувається пошук адреси початку файлу. Результат виконання такого пошуку File і є адресою початку файлу. Тому і тип даних File має відповідний – це покажчик LPVOID.

Взагалі, адресний простір процесу це величезний лінійний масив байтів, в якому зберігається все, з чим безпосередньо працює процес (програма). Масив цей віртуальний, не обмежений фізичною пам'яттю, унікальний для кожної програми і має достатню розмірністю, щоб програміст не замислювався про його обмеження [16].

У другій частині відбувається безпосередня перевірка, чи є файл зараженим? Відповідь на це питання дає порівняння хешів файлу та вірусу.

Більш детально функції, що виконує програма сканеру описано в розділі «Керівництво програміста».

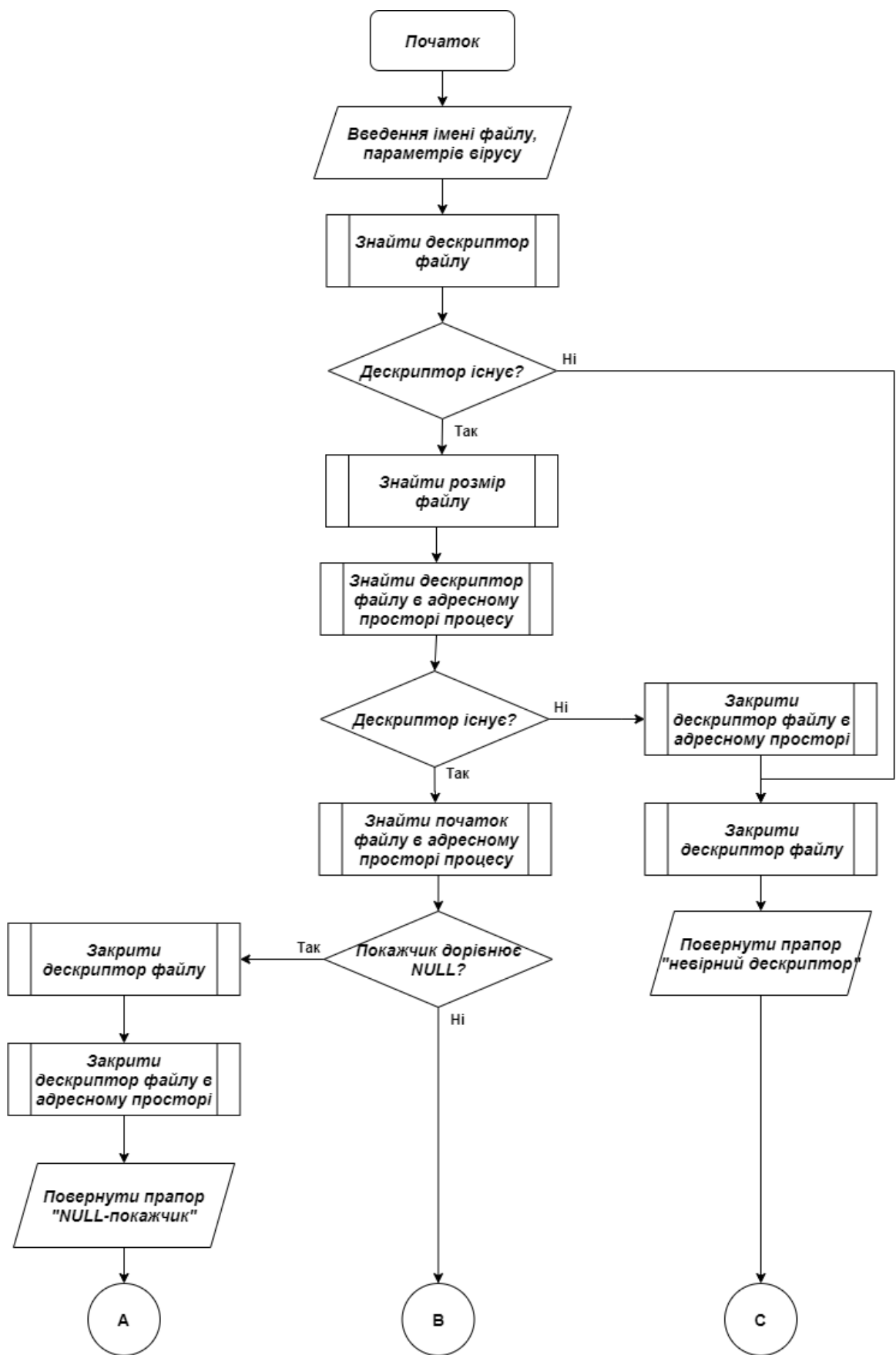


Рисунок 3.1 – Блок-схема алгоритму програми сканера, аркуш 1

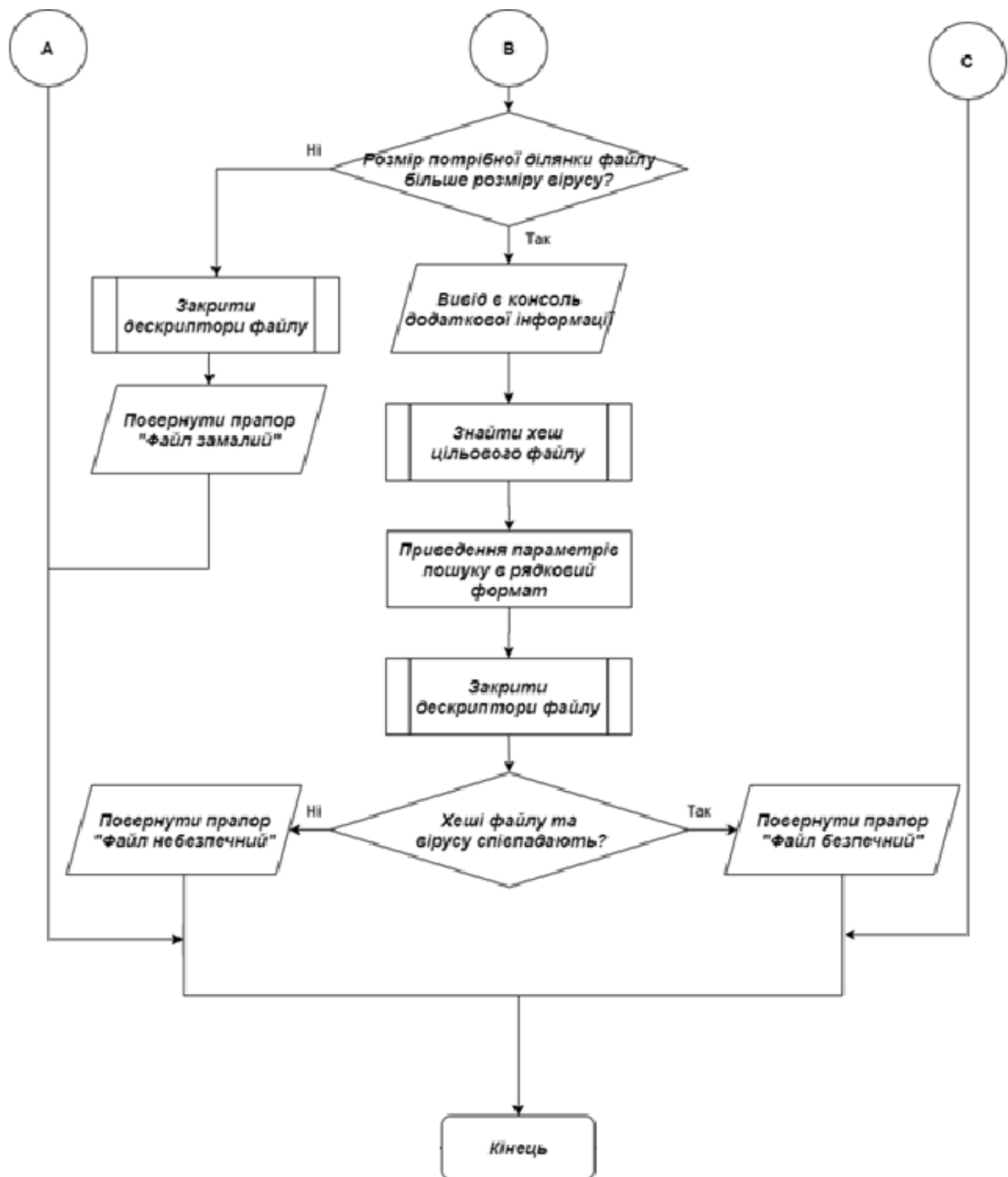


Рисунок 3.1, аркуш 2

3.2 Основні рішення проєкту test.sln

Структура проєкту test.sln складається з трьох класів та одного переліку.

Клас AVRecord описує один конкретний запис про вірус у БД. Розділення такого запису на два класи – AVRecord та AVSignature обумовлено:

- вимогами, висунутими у ТЗ;

– більш чітким розділенням властивостей вірусу (назва та пошукові характеристики окремо);

– відсутність необхідності зберігати розмір імені вірусу, як зазначено у ТЗ. Це відбулося через те, що дані зберігаються не в текстовому документі, а у БД.

Поля та методи класу наведені у табл.3.1.

Таблиця 3.1 – Опис полів та методів класу AVRecord

Поля та методи класу	Опис
<i>public:</i>	
string name	Ім'я вірусу.
AVSignature signature	Характеристики вірусу, що конкретно виділяють його з сукупності інших, та допомагають його виявити
AVRecord()	Конструктор без параметрів, що призначає нульові значення
AVRecord(string name, DWORD offset, DWORD length, const char* hash)	Конструктор, що приймає ім'я вірусу, зміщення послідовності байт, відносно початку, довжину послідовності, рядок з контрольною сумою послідовності в 16-річному форматі.
void ConsoleOutput()	Виводить текстову інформацію в консоль про вірус для розробника.

Клас AVSignature описує сигнатуру вірусу, що зберігається в антивірусній базі. Об'єкт цього класу є обов'язковою частиною об'єкту класу AVRecord. Поля та методи AVSignature наведені у табл. 3.2.

Таблиця 3.2 – Опис полів та методів класу AVSignature

Поля та методи класу	Опис
<i>public:</i>	
DWORD offset	4-байтове зміщення послідовності.
char hash[33]	Контрольна сума вірусу, що виражена у 32 текстових символах 16-річної системи числення.
DWORD length	4-байтовий розмір послідовності.
AVSignature()	Конструктор без параметрів, що призначає нульові значення.
AVSignature(DWORD offset, DWORD length, const char* hash)	Конструктор, що приймає зміщення послідовності байт, відносно початку, довжину послідовності, рядок з контрольною сумою послідовності в 16-річному форматі.
void ConsoleOutput()	Виводить текстову інформацію в консоль про сигнатуру вірусу для розробника.

Структура даних ScannerResult визначає значення, які буде повертати сканер, у тому чи іншому випадку. Тобто сканер повертає не просто логічне «так» чи «ні», у результаті перевірки файлу на наявність вірусу, а значення якогось флагу, що більш точно відбиває результат скану. Поля переїлку описані у табл. 3.3.

Таблиця 3.3 – Значення переліку ScannerResult

Поля переліку	Опис
INVALID_HANDLE	Це значення повертається у тому разі, якщо неможливо отримати дескриптор файлу, тобто його значення є некоректним (invalid). А через те, що не можна посилатися та взаємодіяти з файлом, сканування переривається.
FILE_NULL	Це значення повертається у тому разі, якщо посилання на початок відображення файлу в адресному просторі, викликаного процесу, приймає значення null. А через те, що не можна посилатися та взаємодіяти з файлом, сканування переривається.
SMALL_FILE	Це значення повертається у тому разі, якщо файл замалий для перевірки. Тобто вірус елементарно більше, ніж файл зі зміщенням.
INFECTED_FILE	Це значення повертається у тому разі, якщо файл визначений як «заражений».
HEALTHY_FILE	Це значення повертається у тому разі, якщо файл визначений як «здоровий».

Клас Comments призначений для виводу інформації в консоль, яку має використовувати розробник. Єдиний статичний метод цього класу, ConsoleOut, не повертає значень, та приймає обов'язковий рядковий параметр з повідомленням у консоль, та необов'язковий – з посиланням на рядок. Зазвичай

останній параметр задається, щоб позначити ім'я файлу, з яким на даний момент ведеться робота.

Виконання методу ConsoleOut залежить від визначення чи невизначення константи ConsoleComments. Буде активним цей метод, чи ні, вирішує сам розробник: у разі відсутності необхідності у консольному виводі, константу з директивою define необхідно закоментувати.

Бібліотека CPPScanner.dll створюється в результаті компіляції test.sln, опис структури якого наведено вище. Створена бібліотека динамічного компонування містить ряд функцій, з визначеним порядком зв'язування у стилі C та доступними для використання у інших рішеннях. Готові для експорту методи описані у табл. 3.4, а більш детально у розділі «Керівництво програміста».

Таблиця 3.4 – Опис методів бібліотеки CPPScanner.dll

Методи бібліотеки	Опис
1	2
ForExport bool isFileExist(PCSTR filename)	Визначає, чи існує файл на комп'ютері, за вказаним ім'ям.
ForExport unsigned long FileSize(PCSTR filename)	Визначає розмір файлу у байтах, за вказаним повним шляхом до нього.
ForExport char* FindMD5ForTest(const char * str_for_md5, int n)	Визначає контрольну суму для заданого рядку з визначеною довжиною. Застосовується алгоритм MD5.
ForExport ScannerResult isFileInfected(PCSTR filename, const char* name, DWORD offset, DWORD length, const char* hash, PSTR &handle1, PSTR& handle2, PSTR &FileStartMem, PCSTR& FileHash)	Функція визначає чи є вірус шкідливим. Також повертає додаткові параметри скану, отримані по ходу виконання алгоритму.

Продовження таблиці 3.4

1	2
ForExport ScannerResult isFileInfectedSimple(PCSTR filename, const char* name, DWORD offset, DWORD length, const char* hash)	Спрощений варіант функції вище, без додаткових параметрів. Також визначає, чи є файл вірусом.

3.3 Основні рішення проєкту AV_Manager.sln

Структура проєкту AV_Manager.sln складається з 10 класів (рис. 3.2), з яких:

- один клас – точка входу у програму;
- один клас – з'єднання з функціями CPPScanner.dll;
- один клас – робота з БД та формування запитів;
- три класи – описують сутності з предметної області;
- чотири класи – форми для формування інтерфейсу користувача.

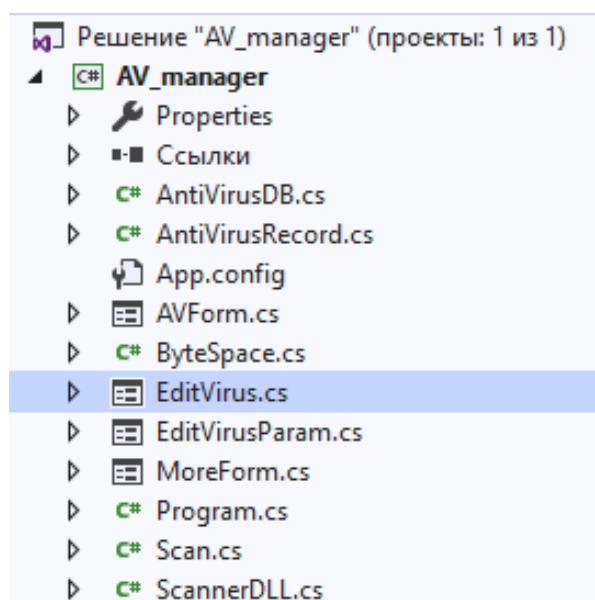


Рисунок 3.2 – Структура класів проєкту AV_Manager

Діаграма класів проєкту AV_Manager наведена на рис. 3.3.

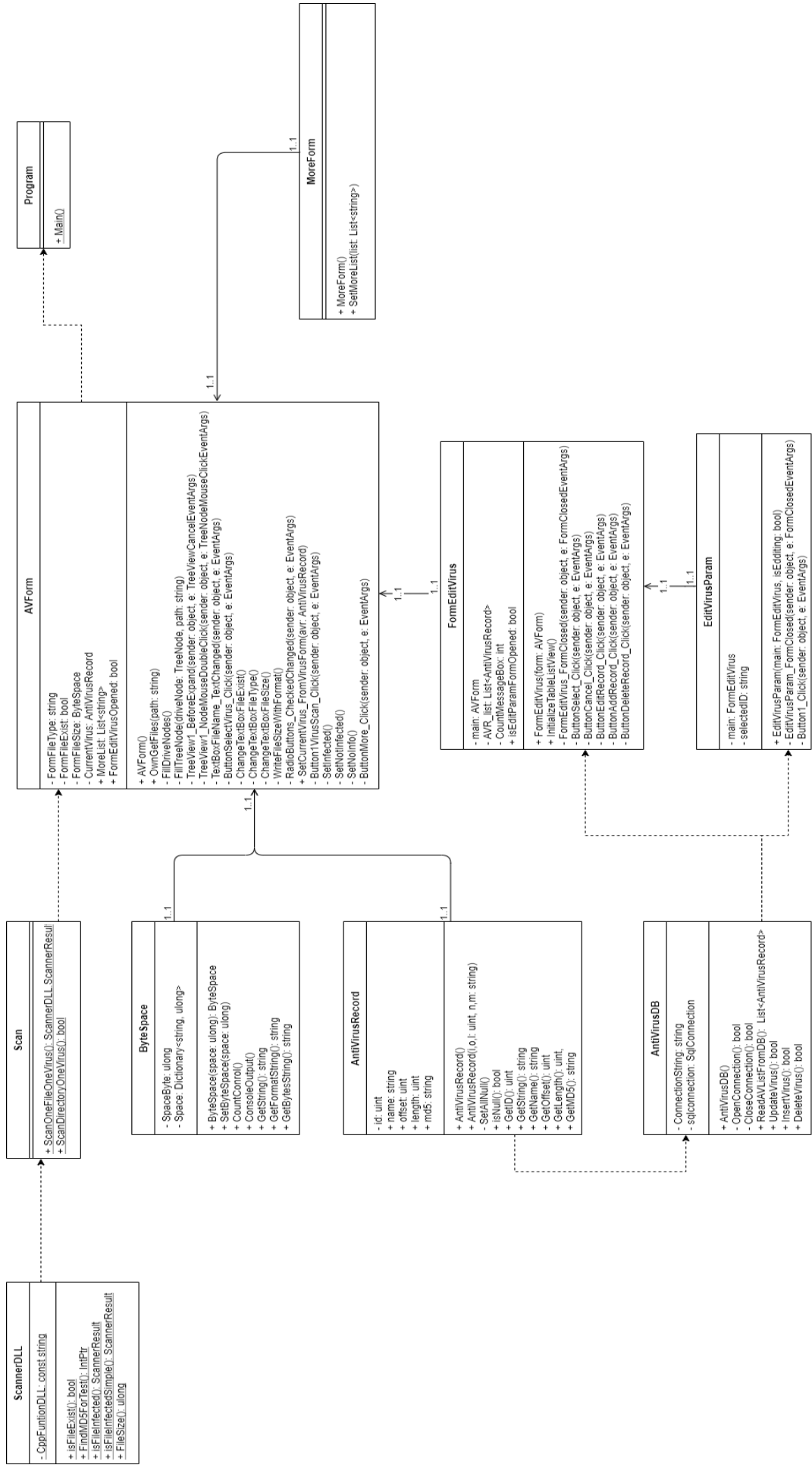


Рисунок 3.3 — Діаграма класів проекту AV_Manager

Клас ScannerDLL необхідний для встановлення зв'язку з розробленою у проєкті test.sln бібліотекою CPPScanner.dll. Це відбувається за допомогою опису сигнатур функцій, з використанням атрибуту DllImport [15]. Через це, структура методів цього класу повністю повторює структуру методів бібліотеки динамічного компонування і виконує ті ж самі задачі. Це стосується і структур даних, що мають бути «схожими» на свої аналоги в C++. Методи та поля класу описані у табл. 3.5.

Таблиця 3.5 – Поля та методи ScannerDLL

Поля класу
private const string CppFunctionDll = @"CPPScanner.dll"
[Flags] public enum ScannerResult
Методи класу
[DllImport(CppFunctionDll, CallingConvention = CallingConvention.Cdecl)] [return: MarshalAs(UnmanagedType.I1)] public static extern bool isFileExist(string filename)
[DllImport(CppFunctionDll, CallingConvention = CallingConvention.Cdecl)] public static extern IntPtr FindMD5ForTest(string str_for_md5, int n)
[DllImport(CppFunctionDll, CallingConvention = CallingConvention.Cdecl)] public static extern ScannerResult isFileInfected(string filename, string name, uint offset, uint length, string hash, out IntPtr handle1, out IntPtr handle2, out IntPtr FileStartMem, out IntPtr FileHash)
[DllImport(CppFunctionDll, CallingConvention = CallingConvention.Cdecl)] public static extern ScannerResult isFileInfectedSimple(string filename, string name, uint offset, uint length, string hash)
[DllImport(CppFunctionDll, CallingConvention = CallingConvention.Cdecl)] public static extern ulong FileSize(string filename)

Клас Scan визначає функції, які безпосередньо будуть використані програмою, для визначення шкідливості файлу. Ці функції виконують ряд

проміжних дій, які допомагають більш повно проаналізувати файл або директорію на наявність вірусу. Клас Scan нерозривно зв'язаний з функціоналом класу ScannerDLL. Методи класу описані у табл. 3.6.

Таблиця 3.6 – Методи класу Scan

Методи класу	Опис
<i>public:</i>	
public static ScannerDLL.ScannerResult ScanOneFileOneVirus(string filename, AntiVirusRecord avr, out string h1, out string h2, out string FSM, out string FH)	Функція реалізує класичну перевірку конкретного файлу на наявність одного конкретного вірусу. З результатом такої перевірки, функція також повертає і додаткові дані про дескриптори файлу, про адрес початку файлу, а також про його контрольну суму.
public static bool ScanDirectoryOneVirus(string filename, AntiVirusRecord avr, out List<string> ResultList)	Функція реалізує перевірку конкретної директорії на наявність одного конкретного вірусу. Додатково, функція повертає лог перевірки: список з ім'ям вірусу, ім'ям файлу директорії, результатом цієї попарної перевірки.

Клас AntiVirusDB забезпечує доступ та встановлення зв'язку з БД, а також експорт та імпорт даних. База даних створена за допомогою SQL Server, відповідно запити написані мовою SQL. Поля та методи класу описані у табл. 3.7.

Таблиця 3.7 – Поля та методи класу AntiVirusDB

Методи класу	Опис
1	2
public:	
private static string ConnectionString = "Data Source = (LocalDB)\\MSSQLLocalDB;AttachDb Filename=" + Environment.CurrentDirectory + "\\Anti-VirusDB.mdf;Integrated Security = True";	Рядок підключення, або «connection string». Визначає місце розташування файлу БД, для встановлення підключення та подальшої роботи. Шлях до файлу – відносний. Цього було досягнуто завдяки застосуванню змінної Environment.CurrentDirectory.
private SqlConnection sqlconnection	Об'єкт класу SqlConnection, що визначає підключення до БД.
private bool OpenConnection()	Метод, що в разі, якщо підключення до БД ще не відбулось, відкриває його, тим самим забезпечуючи доступ до даних БД.
private bool CloseConnection()	Метод, що в разі, якщо підключення до БД відкрите, закриває його, тим самим обмежуючи доступ до неї.
public AntiVirusDB()	Відкритий конструктор, що ініціалізує об'єкт бази даних, необхідний для взаємодії з нею. Іншими словами, конструктор ініціалізує поле sqlconnection рядком з'єднання з поля ConnectionString.
public List<AntiVirusRecord> ReadAVListFromDB()	Формує запит до БД, та повертає усі відомі записи про віруси. Ці записи організовані у список.

Продовження таблиці 3.7

1	2
public bool UpdateVirus(string id, string name, string offset, string len, string md5)	Формує запит до БД, та змінює параметри у 1 визначеного вірусу за його Id. Повертає результат успіху зміни параметрів.
public bool InsertVirus(string name, string offset, string len, string md5)	Формує запит до БД, та додає до неї запис про вірус, з визначеними параметрами. Повертає результат успіху додавання вірусу.
public bool DeleteVirus(string id)	Формує запит до БД, та видаляє визначений вірус за його Id. Повертає результат успіху видалення вірусу.

Клас AntiVirusRecord є аналогом структури класів AVRecord та AVSignature, що були розроблені у проекті test.sln, написаному на C++. Точно таким же чином, як і його «родичі», клас AntiVirusRecord визначає, поняття «вірус» в системі, які параметри та методи він має. Тут варто зауважити, що властивості, замість змінних, не використовувались. Об'єкт цього класу безпосередньо взаємодіє з класом AntiVirusDB, та, так би мовити, є цеглинкою усієї антивірусної бази. Поля та методи класу описані у табл. 3.8.

Таблиця 3.8 – Структура полів та методів AntiVirusRecord

Поля та методи класу	Опис
1	2
private:	
private uint id	Поле, що визначає Id вірусу в антивірусній базі.

Продовження таблиці 3.8

1	2
private void SetAllNull()	Метод, що призначає полям класу їх значення за замовчуванням, або нульові значення. Якщо ці значення активні, то увесь вірус вважається нульовим, тобто не дійсним
public:	
public string name	Поле, що визначає ім'я вірусу в антивірусній базі.
public uint offset	Поле, що визначає зміщення послідовності байт для пошуку вірусу.
public uint length	Поле, що визначає довжину послідовності байт для пошуку вірусу.
public string md5	Поле, що визначає контрольну суму вірусу в антивірусній базі.
public AntiVirusRecord()	Конструктор без параметрів, що визиває метод SetAllNull().
public AntiVirusRecord(uint i, string n, uint o, uint l, string m)	Конструктор з параметрами вірусу. Конструктор спочатку встановлює нульові значення, визиваючи метод SetAllNull(). Після цього перевіряє формат параметрів. У разі їх коректності, з'являється повноцінний об'єкт вірусу з заданими параметрами. У іншому випадку, об'єкт вірусу є нульовим.
public bool isNull()	Перевіряє, чи є вірус нульовим, тобто не дійсним, та неготовим до роботи. Такий вірус має значення Id = 0.
public string GetName()	Метод повертає значення поля name об'єкту вірусу.

Продовження таблиці 3.8

1	2
public uint GetID()	Метод повертає значення поля id об'єкту вірусу.
public uint GetOffset()	Метод повертає значення поля offset об'єкту вірусу.
public uint GetLength()	Метод повертає значення поля length об'єкту вірусу.
public string GetMD5()	Метод повертає значення поля md5 об'єкту вірусу.
public string GetString()	Метод формує зі всієї інформації про вірус цільовий рядок та повертає його. Функцію доцільно використовувати для тестування, щоб швидко отримати усі необхідні значення полів.

Об'єкт класу ByteSpace зберігає кількість байт, отриману, наприклад, при вимірі розміру файлу. Перевага цього класу в тому, що він подає байти як в розгорнутому (повному) форматі, так і в звичному для користувача – зі зрозумілими одиницями виміру: Кб, Мб і т.д. Поля та методи класу описані у табл. 3.9.

Таблиця 3.9 – Поля та методи класу ByteSpace

Поля та методи класу	Опис
1	2
private:	
ulong SpaceByte	Поле, що зберігає кількість байт у звичайному десятинному форматі.
Dictionary<string, ulong> Space	Поле, представлене словником. Ключами виступають рядки: "B", "KB", "MB", "GB", "TB", а значеннями – розмір кожного виміру.

Продовження таблиці 3.9

<i>public:</i>	
public ByteSpace(ulong space)	Метод, що ініціалізує об'єкт заданою кількістю байт. Одразу відбувається перевід байт в формат під поле Space.
public void SetByteSpace(ulong space)	Метод, що змінює значення об'єкту заданою кількістю байт. Одразу відбувається перевід байт в формат під поле Space.
public void CountConrol()	Дуже важлива функція на етапі тестування. Перевіряє, чи правильно було переведено в формат Space. А точніше, чи дорівнює кількість байт SpaceByte відформатованої кількості байт Space. Результат перевірки виводиться у консоль.
public void ConsoleOutput()	Метод для тестування. Виводить значення об'єкту класу у двох можливих форматах. Вивід також організовано у консоль, де можна порівняти та подивитись на розмір в байтах.
public string GetString()	Метод, що повертає рядок, з повним зазначенням кількості, в розмірі SpaceByte, байт, кілобайт, мегабайт і т.д.
public string GetFormatString()	Метод повертає відформатований рядок Space. Приклад виводу: 32,6 MB.
public string GetBytesString()	Метод повертає невідформатований рядок SpaceByte. Приклад виводу: 321675 – це значення байт.

Клас AVForm представляє звичайний опис об'єкту користувацької форми. Однак нам принципово знати не тільки опис елементів форми та її подій, а й ті поля, які має об'єкт класу AVForm. Значення цих полів, впливають на хід проведення сканування. Поля та методи класу описані у табл.і 3.10.

Таблиця 3.10 – Поля та методи класу AVForm

Поля та методи класу	Опис
1	2
<i>private:</i>	
private static string FormFileType	Поле, що визначає тип поточного обраного файлу форми.
private static bool FormFileExist	Поле, що визначає, чи існує поточний обраний файл форми.
private static ByteSpace FormFileSize	Поле, що визначає розмір поточного обраного файлу форми. Для цього застосовується користувацький тип даних ByteSpace.
private static AntiVirusRecord CurrentVirus	Поле, що визначає, поточний обраний вірус форми.
private void FillDriveNodes()	Метод, що отримує інформацію про диски системи, та завантажує їх у вигляді вузлів елементу treeView.
<i>public:</i>	
public static List<string> MoreList	Поле, що зберігає список, отриманий в результаті аналізу директорії на вміст одного конкретного вірусу. Список складається з рядків з інформацією про назву файлу, вірусу та результат перевірки.

Продовження таблиці 3.10

1	2
public bool FormEditVirusOpened	Поле, що вказує, чи відкрите вікно редагування антивірусної бази (форма EditVirus). Поле потрібне для відображення форми лише у одному вікні.
public AVForm()	Конструктор, який ініціалізує дані форми.
public static string[] OwnGetFiles(string path)	Метод, що повертає усі файли, що зберігаються за вказаною адресою (шляхом) для заданої директорії.
public void SetCurrentVirus_FromVirusForm(AntiVirusRecord avr)	Метод, що призначений для виклику з форми EditVirus. Він встановлює значення поточного вірусу, відповідно переданого параметру. Цей параметр – це вірус, вибраний користувачем зі всіх можливих у формі EditVirus.
private:	
private void FillTreeNode(TreeNode driveNode, string path)	Метод, що заповнює treeView вузлами – директоріями, які можна розгорнути та в свою чергу продивитися їх вміст.
private void TreeView1_BeforeExpand(object sender, TreeViewCancelEventArgs e)	Подія, що відбувається у разі розгортання вузла елемента treeView. Після цього для розгорнутого вузлу будується своя власна структура з файлами та каталогами.

Продовження таблиці 3.10

1	2
private void TextBoxFileName_TextChanged(object sender, EventArgs e)	Подія, що відбувається у разі зміни тексту в текстовому полі цільового файлу. Вона викликає послідовне виконання трьох функцій: ChangeTextBoxFileExist(), ChangeTextBoxFileType(), ChangeTextBoxFileSize().
private void ButtonSelectVirus_Click(object sender, EventArgs e)	Подія, що відбувається у разі натискання на кнопку «Вибрати вірус», позначену трьома крапками. У цьому разі буде створена та відображена форма EditVirus.
private void ChangeTextBoxFileExist()	Метод, що аналізує поточний файл форми, та в залежності від того, існує він чи ні, змінює форматування відповідного текстового поля.
private void ChangeTextBoxFileType()	Метод, що аналізує поточний файл форми, та в залежності від його типу, звичайний файл або директорія, змінює форматування відповідного текстового поля.
private void ChangeTextBoxFileSize()	Метод, що аналізує поточний файл форми, та від його розміру в байтах, змінює форматування відповідного текстового поля.
private void WriteFileSizeWithFormat()	Метод, що аналізує стан елементів radioButton, та в залежності від вибору користувача, змінює форматування розміру файлу (див. клас ByteSpace).

Продовження таблиці 3.10

1	2
private void RadioButtons_CheckedChanged(object sender, EventArgs e)	Подія, що відбувається у разі зміни користувачем стану перемикача radioButton, та в залежності від його вибору, змінює форматування розміру файлу (див. клас ByteSpace).
private void Button1VirusScan_Click(object sender, EventArgs e)	Подія, що відбувається у разі натискання на кнопку сканування. Програма аналізує поточні поля системи (про файл та вірус), та згідно до них визиває потрібний метод сканування, передаючи туди ці параметри. Після сканування ця функція встановлює форматування необхідних полів, що відбивають результат сканування.
private void SetInfected()	Метод, що встановлює відповідне форматування поля з результатом сканування – «файл небезпечний».
private void SetNotInfected()	Метод, що встановлює відповідне форматування поля з результатом сканування – «файл безпечний».
private void SetNoInfo()	Встановлює відповідне форматування поля з результатом сканування – «результат невідомий». Такий сценарій відбувається або у разі сканування директорії, або у разі помилки сканування.

Продовження таблиці 3.10

1	2
<code>private void ButtonMore_Click(object sender, EventArgs e)</code>	Подія, що відбувається у разі натискання на кнопку відображення логу «More». В такому разі створюється та відображається форма <code>MoreForm</code> з результатом сканування директорії.

Клас `FormEditVirus` представляє звичайний опис об'єкту користувацької форми. Таку назву форма отримала через свої головні обов'язки – забезпечення взаємодії користувача з антивірусною базою та вибір цільового вірусу для сканування для головної форми. Поля та методи класу описані у табл. 3.11.

Таблиця 3.11 – Поля та методи класу `FormEditVirus`

Поля та методи класу	Опис
1	2
<i>private:</i>	
<code>AVForm main</code>	Поле, що зберігає посилання на головну форму програми. Тому важливо правильно обмежити рівні доступу у класі <code>AVForm</code> .
<code>private static int CountMessageBox</code>	Поле, що зберігає кількість разів, яке користувач бачив повідомлення (не помилку). Для того, щоб не перенавантажувати увагу користувача, необов'язкові та малозначущі повідомлення з'являються лише один раз.

Продовження таблиці 3.11

1	2
List<AntiVirusRecord> AVR_list	Поле, що зберігає список завантажених із БД вірусів. Зі зміною даних в БД цей список повинен поновлюватися.
public:	
public bool isEditParamFormOpened	Поле, що вказує, чи відкрите вікно редагування запису антивірусної бази (форма EditVirusParam). Поле потрібне для відображення форми лише у одному примірнику.
public FormEditVirus(AVForm form)	Конструктор, який ініціалізує дані форми.
public void InitializeTableListView()	Метод, що ініціалізує та заповнює даними елемент TableListView, який відображає усі віруси, відомі антивірусній базі.
private :	
private void FormEditVirus_FormClosed(object sender, FormClosedEventArgs e)	Подія, що відбувається у разі закриття форми FormEditVirus. У такому разі головній формі відправляємо сигнал, про закриття поточної. Там змінюється значення поля FormEditVirusOpened на false.
private void ButtonSelect_Click(object sender, EventArgs e)	Подія, що відбувається у разі натискання на кнопку «Вибрати». У такому разі для головної форми виконається функція SetCurrentVirus_FromVirusForm() з обраним користувачем вірусом.

Продовження таблиці 3.11

1	2
private void ButtonCancel_Click(object sender, EventArgs e)	Подія, що відбувається у разі натискання кнопки «Відмінити». Поточна форма буде закрита.
private void ButtonEditRecord_Click(object sender, EventArgs e)	Подія, що відбувається у разі натискання кнопки «Редагувати». Буде створено об'єкт форми EditVirusParam з параметром true.
private void ButtonAddRecord_Click(object sender, EventArgs e)	Подія, що відбувається у разі натискання кнопки «Додати». Буде створено об'єкт форми EditVirusParam з параметром false .
private void ButtonDeleteRecord_Click(object sender, EventArgs e)	Подія, що відбувається у разі натискання користувачем кнопки «Видалити». Обраний із списку вірус буде видалений з БД, і у разі успіху зі списку AVR_list.

Клас EditVirusParam представляє звичайний опис об'єкту користувацької форми. Хоч структура цієї форми здається невеликою, вона відповідає за дві важливі функції системи: додавання та редагування записів у антивірусній базі. Поля та методи класу описані у табл. 3.12.

Таблиця 3.12 – Поля та методи класу EditVirusParam

Поля та методи класу	Опис
1	2
private:	
FormEditVirus main;	Посилання на батьківську форму (не головну).

Продовження таблиці 3.12

1	2
<pre>private static string selectedID = ""</pre>	Поле, що вказує, який індекс у БД має поточний, обраний у списку на формі FormEditVirus, вірус. Якщо форма відкрита для додавання нового вірусу, поле selectedID залишається порожнім.
<pre>private void EditVirusParam_FormClosed(object sender, FormClosedEventArgs e)</pre>	Подія, що відбувається у разі закриття форми EditVirusParam. У такому разі батьківській формі відправляємо сигнал про закриття поточної. Там змінюється значення поля isEditParamFormOpened на false.
<pre>private void Button1_Click(object sender, EventArgs e)</pre>	Подія, що відбувається у разі натискання на кнопку «Додати». Водночас тут відбуваються головні алгоритмічні операції по редагуванню або додаванню у систему вірусу. однак схема для цих двох процесів схожа: – визначення однієї з двох функцій для виконання; – зміна запису про вірус у БД; – в разі невдачі вивід повідомлення про неможливість операції; – в разі успіху – зміна запису про вірус у програмі (у списку AV_list та у списку viewList).

Продовження таблиці 3.12

1	2
<i>public:</i>	
public EditVirusParam(FormEdit Virus main, bool isEdditing)	Конструктор, який ініціалізує дані форми. Звичайно він приймає посилання на батьківську форму та параметр isEdditing: – у разі, якщо його значення істина, форма застосовується для редагування вже існуючого вірусу. Поля параметрів заповнюються тими значеннями, які є у того вірусу, що вибрав користувач у списку з батьківської форми; – у разі, якщо його значення помилкове, форма застосовується для створення принципово нового запису про вірус. Поля параметрів вірусу порожні, їх повинен заповнити сам користувач.

Проста форма MoreForm зберігає елемент listBox, який за потреби заповнюється даними про результат множинного сканування, тобто сканування директорії. Поля та методи класу MoreForm описані у табл. 3.13.

Таблиця 3.13 – Поля та методи класу MoreForm

Поля та методи класу	Опис
<i>public:</i>	
public MoreForm()	Конструктор, який ініціалізує дані форми.
public void SetMoreList(List<string> list)	Заповнює список форми даними, які передаються в якості параметру. Зазвичай це список, створений в результаті множинного сканування.

3.4 Особливості розробки програми антивірусу

Даний антивірус був розроблений за допомогою середовища розробки Microsoft Visual Studio 19. Ніякої принципової різниці, між роботою в аналогічному середовищі 2017 року і в версії 2019 немає: точно такий же необхідний функціонал для проєкту є у обох випадках.

Щодо створення бібліотеки динамічного компонування засобами мови C#, то в обох версіях Visual Studio можна обрати спеціальний тип проєкту – бібліотека класів. Приклад створення такого проєкту зображено на рис. 3.4.

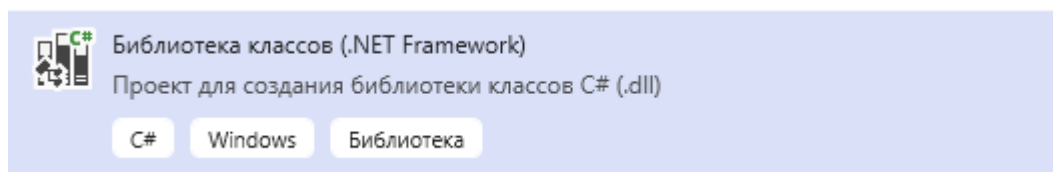


Рисунок 3.4 – Створення dll бібліотеки на мові C#

Однак для створення такої бібліотеки мовою C++ виникли труднощі – у Microsoft Visual Studio 17 немає відповідного пункту, точніше він не був знайдений у Visual Studio Installer для додаткового завантаження. Робота з Visual Studio Installer зображена на рис. 3.5.

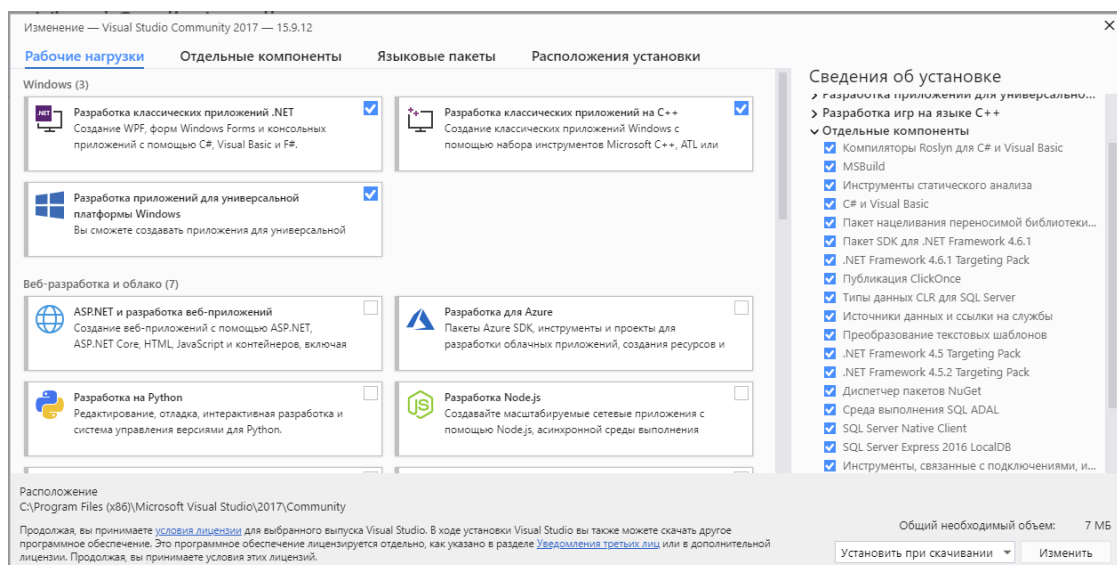


Рисунок 3.5 – Робота з Visual Studio Installer

Через це, було прийнято рішення розробляти проєкт засобами Microsoft Visual Studio 19. Створення проєкту бібліотеки динамічного компонування здійснювалась наступним чином. Спочатку було створено порожній проєкт. Цей процес зображений на рис. 3.6.

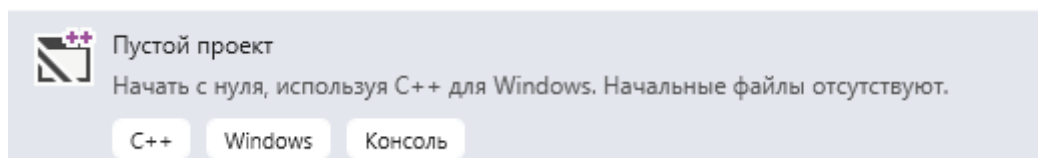


Рисунок 3.6 – Відповідний пункт для створення порожнього проєкту

Зміна у вікні налаштувань проєкту типу конфігурації: з .exe на .dll. Подібна опція зображена на рис. 3.7.

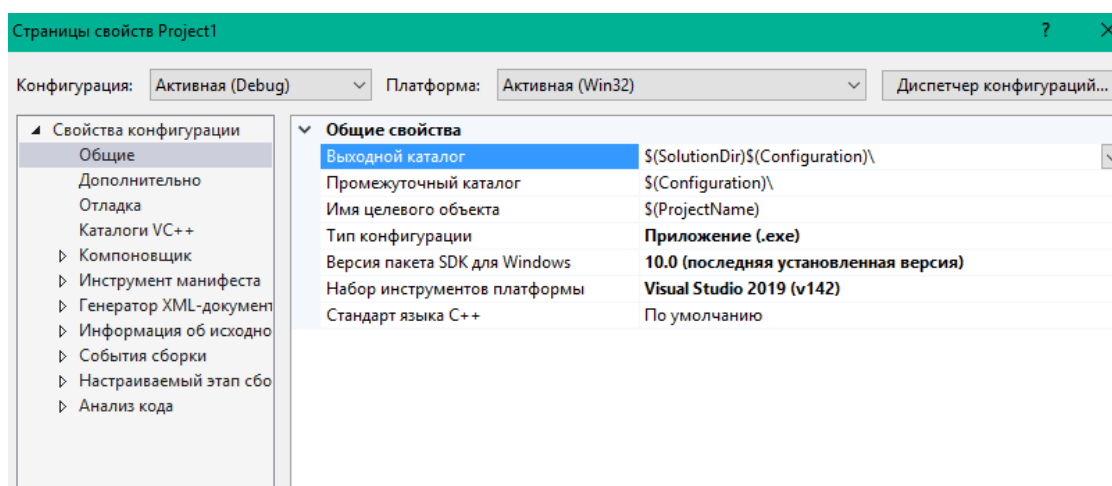


Рисунок 3.7 – Зміна типу конфігурації проєкту з .exe на .dll

Також варто зазначити, що подібна зміна типу конфігурації проєкту можлива і у версії 2017 року. Однак, якщо у Visual Studio 19 інші налаштування проєкту реально дозволяють надалі створювати dll після такої зміни, то у версії 17 необхідно задавати ще низку параметрів проєкту.

Також варто зазначити, що є ще один спосіб створення dll мовою C++. Створення відповідного проєкту у середовищі Microsoft Visual Studio зображено на рис. 3.8.

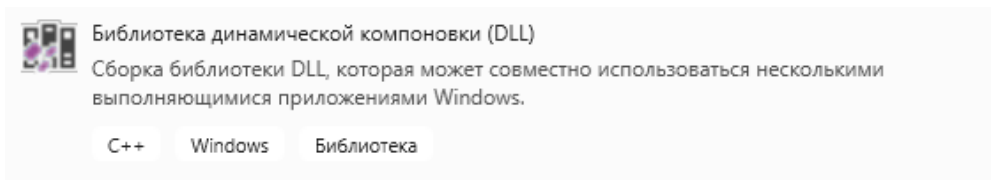


Рисунок 3.8 – Створення проєкту .dll у Microsoft Visual Studio 19

Саме можливість створення такого проєкту відсутня у версії 2017 року, тому доцільніше перейти на більш пізнішу версію – версію Visual Studio 19.

3.5 Розробка основних функцій сканування. Логіка пошуку вірусу

В загальному вигляді функція сканування забезпечує перевірку одного обраного файлу на наявність одного визначеного вірусу. Програмно вона виконана у вигляді функції `isFileInfected`, схема алгоритму якої має вигляд, зображений на рис. 3.9.

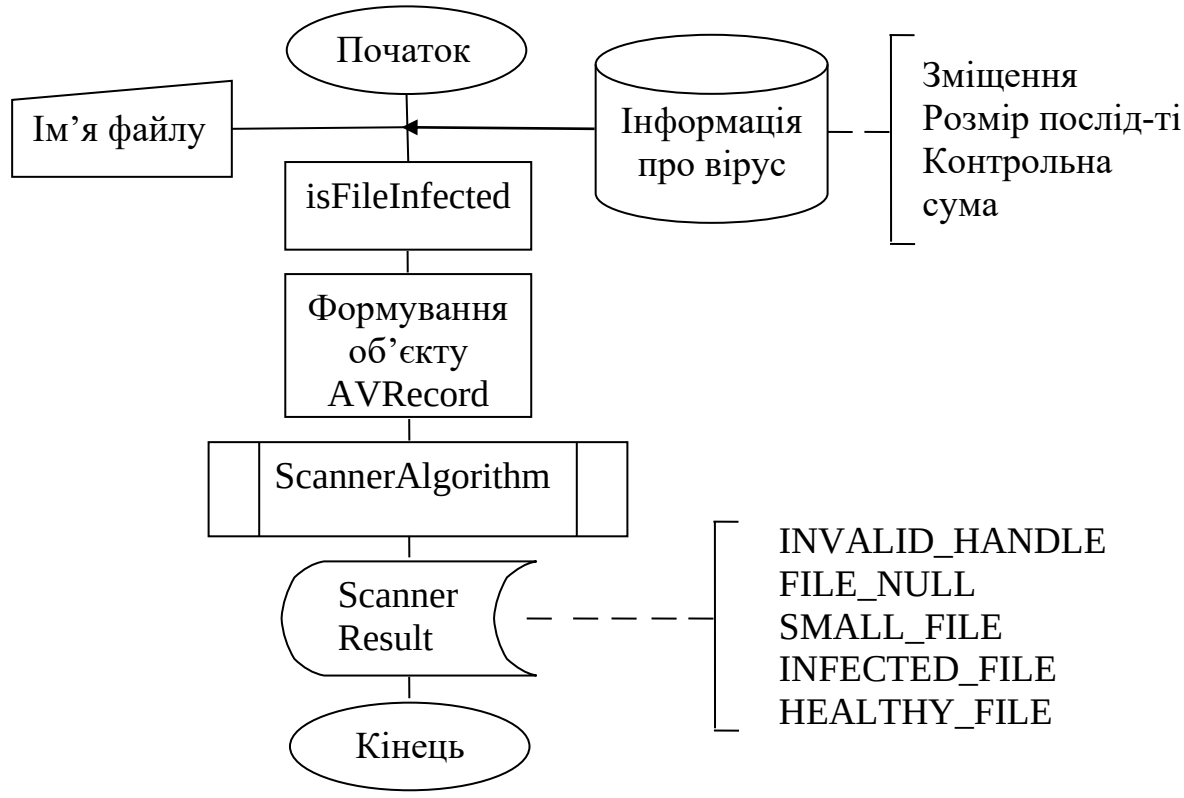


Рисунок 3.9 – Схема алгоритму функції `isFileInfected`

Результат сканування подається у вигляді переліку, що вже зазначався.

Інформація про вірус містить:

- зміщення послідовності байт у файлі, де потрібно шукати вірус. Це, так би мовити, додатковий параметр, що значно спрощує та зменшує час виконання пошуку. За його відсутності необхідно б було проглядати кожен з варіантів розташування вірусу повним перебором. Однак у використанні зміщення є один дуже суттєвий мінус: щоб "обійти" сканер, досить злегка "пересунути" сукупність даних у файлі, тобто змінити зміщення послідовності;
- розмір послідовності байт у файлі. Цей параметр визначає кількість байт, яка буде порівнюватися з байтами вірусу. Тобто фактично, цей розмір і є розміром вірусу;
- контрольна сума, або хеш. Порівнювати байти між собою було б досить проблематично, тому будемо порівнювати їх контрольні суми. Контрольні суми визначаються хеш функціями, що дозволяють однозначно, без повторень, закодувати інформацію. Тому іншого такого коду, як у визначеній послідовності байт не існують. Цікаво, що хеш функція для заданої інформації теж повертає послідовність байт, однак фіксовану та незрівнянно малу – 16 байт. Ці 16 байт прийнято представляти у вигляді 32 символьного рядка 16-річної системи числення. Тобто один байт представлений двома символами.

Для обчислення контрольної суми застосовано популярний алгоритм MD5. Таку структуру даних можна назвати сигнатурою вірусу. В інтернеті існує досить багато джерел, з сигнатурами [17]. На рис. 3.10 зображена одна з таких сигнатур.

Тепер докладніше розглянемо виконання функції `ScannerAlgorithm`. Умовно її алгоритм можна поділити на дві частини:

- пошук, в адресному просторі процесу, початку файлу, що перевіряється;
- визначення за знайденим покажчиком контрольної суми необхідних байт файлу, та перевірка її з хешем вірусу. Повернення результату.

VirusShare.com - Because Sharing is Caring

Home • Hashes • Research • About

Please [login](#) to search and download.

System currently contains 36,718,200 malware samples.

Report for a sample recently added to the system:
508cd3d530af3a57ff5eb5ba61b07d436e792da6d72c4dabe7818c251bb52cdd

VirusShare info last updated 2020-12-22 00:01:03 UTC



MD5	e672a52267090c808e2539815fce2c23																		
SHA1	0b33accb74dc0964520d9309233025d30b4a66e1																		
SHA256	508cd3d530af3a57ff5eb5ba61b07d436e792da6d72c4dabe7818c251bb52cdd																		
SSDeep	3072:6Q4SPZD3UcjvG8rM2hFcXmNRSFI4t+rBKMLZosBbt2HrCFY0ZRxiYHaxkB:IJwXmNRKBFP																		
Size	197,845 bytes																		
File Type	HTML document, UTF-8 Unicode text, with very long lines																		
Mime Type	text/html																		
Extension	html																		
TrID	Atom web feed (92.2%) HyperText Markup Language (7.7%)																		
Detections (2/62)	<table><tr><td>Fortinet</td><td>JS/ScriptP.INFtr</td></tr><tr><td>Microsoft</td><td>Trojan.Script/Wacatac.B!ml</td></tr></table> VirusTotal Report submitted 2020-11-30 14:19:19 UTC			Fortinet	JS/ScriptP.INFtr	Microsoft	Trojan.Script/Wacatac.B!ml												
Fortinet	JS/ScriptP.INFtr																		
Microsoft	Trojan.Script/Wacatac.B!ml																		
ExIF Data	<table><tr><td>ContentType</td><td>text/html; charset=UTF-8</td></tr><tr><td>FileSize</td><td>193 kB</td></tr><tr><td>FileType</td><td>HTML</td></tr><tr><td>FileTypeExtension</td><td>html</td></tr><tr><td>Generator</td><td>blogger</td></tr><tr><td>MIMEType</td><td>text/html</td></tr><tr><td>Title</td><td>Feliz Natal! ~ Taverna do Elfo e do Arcanios</td></tr><tr><td>viewport</td><td>width=device-width, initial-scale=1.0</td></tr></table>			ContentType	text/html; charset=UTF-8	FileSize	193 kB	FileType	HTML	FileTypeExtension	html	Generator	blogger	MIMEType	text/html	Title	Feliz Natal! ~ Taverna do Elfo e do Arcanios	viewport	width=device-width, initial-scale=1.0
ContentType	text/html; charset=UTF-8																		
FileSize	193 kB																		
FileType	HTML																		
FileTypeExtension	html																		
Generator	blogger																		
MIMEType	text/html																		
Title	Feliz Natal! ~ Taverna do Elfo e do Arcanios																		
viewport	width=device-width, initial-scale=1.0																		

Рисунок 3.10 – Докладний опис одного з вірусів

Тепер докладніше розглянемо виконання функції `ScannerAlgorithm`. Умовно її алгоритм можна поділити на 2 частини:

- пошук, в адресному просторі процесу, початку файлу, що необхідно перевірити;
- визначення за знайденим показником контрольну сума необхідних байт файлу, та перевірка її з хешем вірусу. Повернення результату.

Блок-схема алгоритму цієї функції наведена на рис. 3.11.

Кілька слів про першу частину. У ній відбувається пошук початку файлу в адресному просторі процесу, що викликаний. Тобто відбувається пошук адреси початку файлу. Результат виконання такого пошуку `File i` є адресою початку файлу. Тому `i` тип даних `File` має відповідний показник – `LPVOID`.

Взагалі, адресний простір процесу це величезний лінійний масив байтів, в якому зберігається все, з чим безпосередньо працює процес (програма).

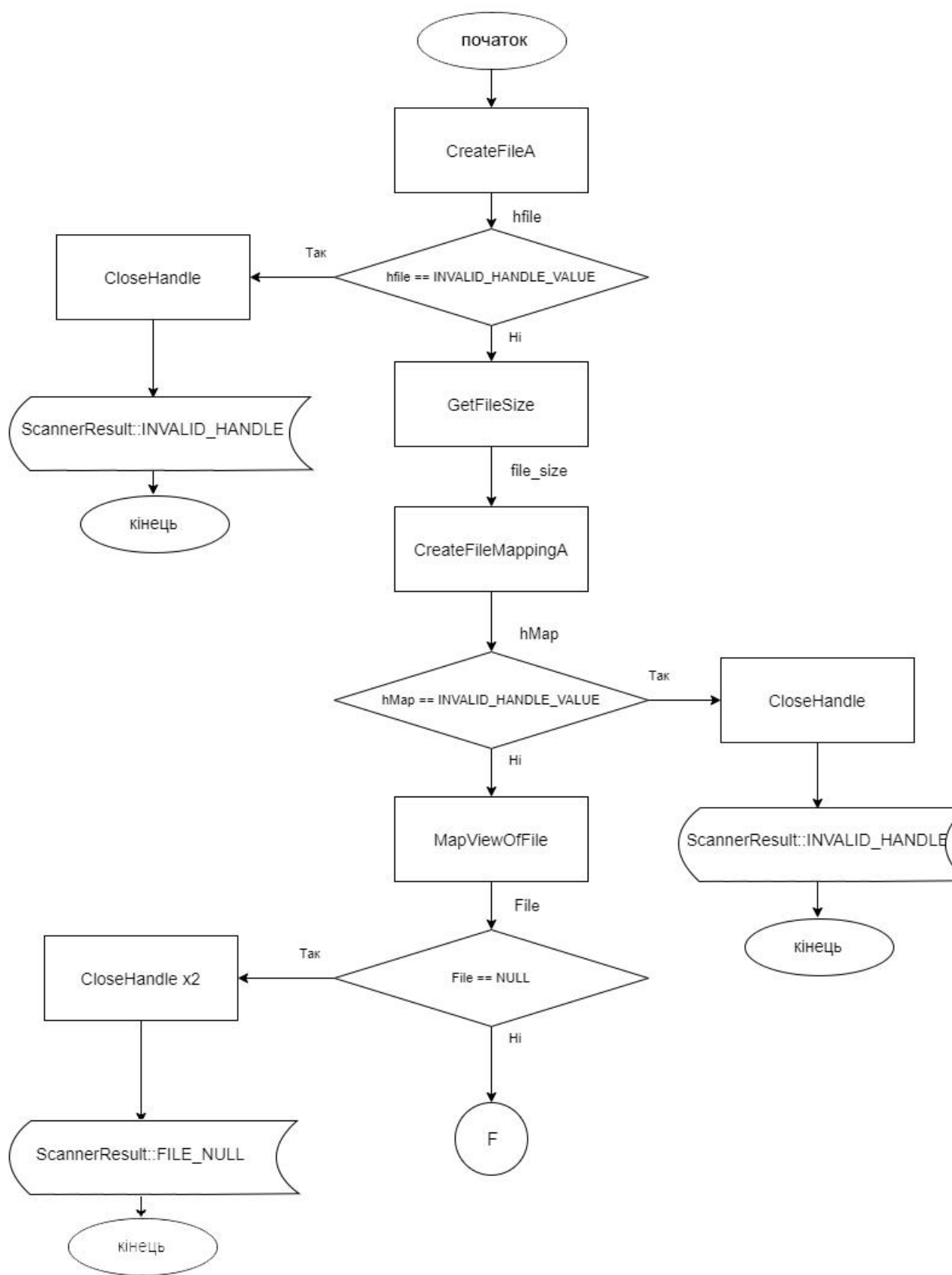


Рисунок 3.11– Блок-схема алгоритму функції ScannerAlgorithm, аркуш 1

Масив цей віртуальний, не обмежений фізичною пам'яттю, унікальний для кожної програми і має достатню розмірність, щоб програміст не замислювався про його обмеження.

Тепер перейдемо до другої частини функції ScannerAlgorithm, яка наведена на рис. 3.11, аркуш 2.

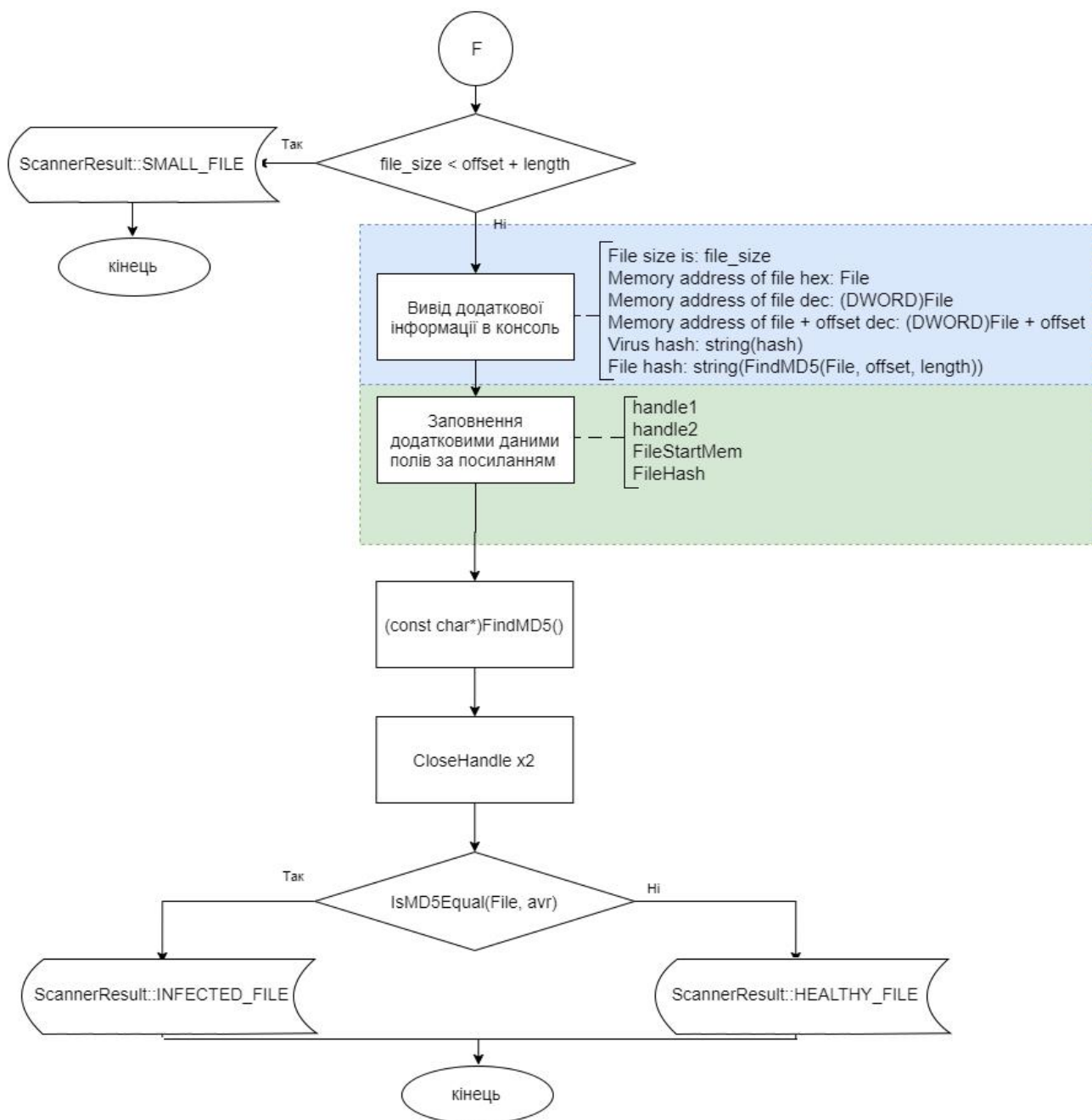


Рисунок 3.11, аркуш 2

Основною тут є функція IsMD5Equal, що знаходить контрольну суму файлу та звіряє її з вірусом. Блок-схема алгоритму функції IsMD5Equal наведена на рис. 3.12.

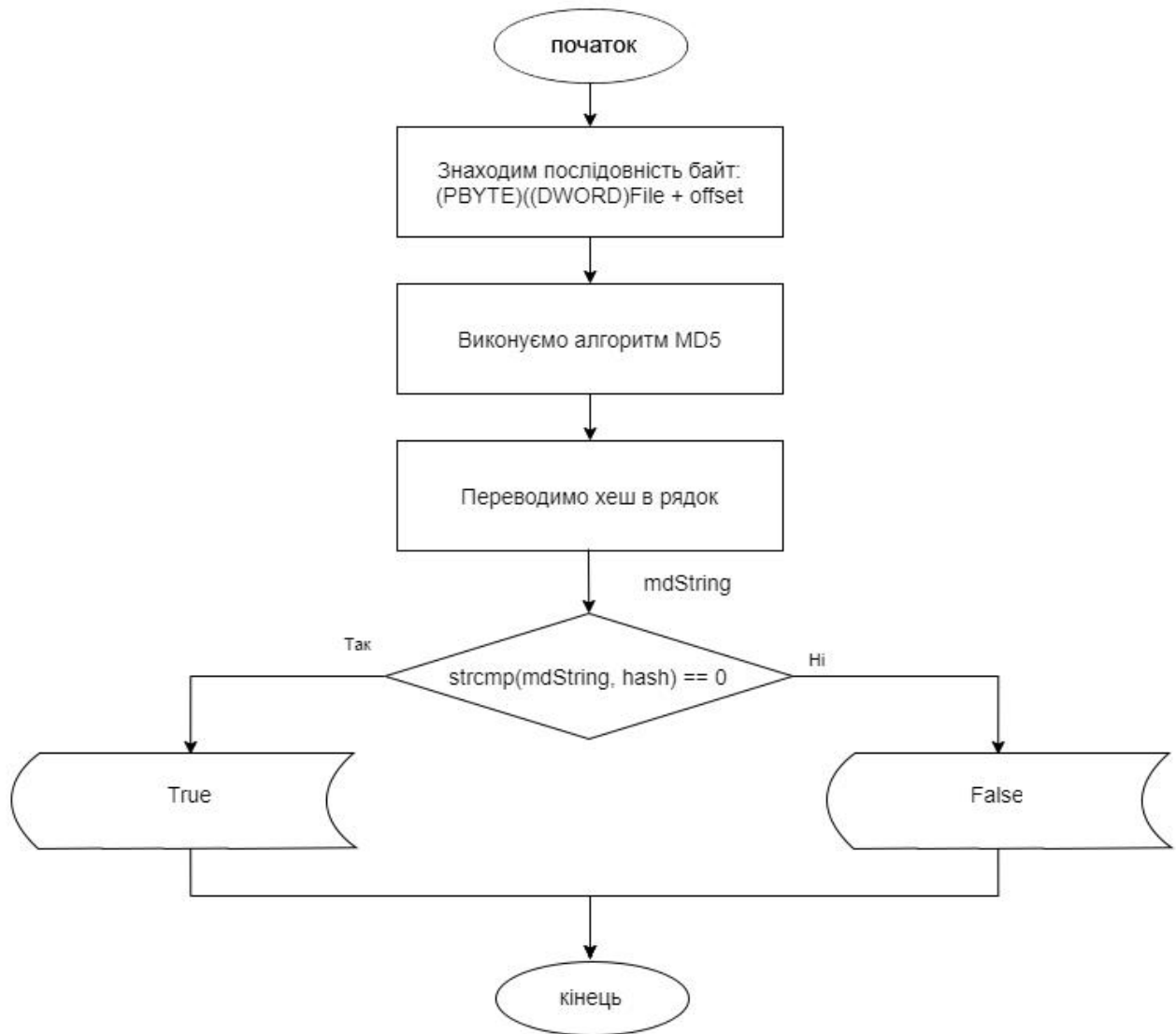


Рисунок 3.12 – Блок-схема алгоритму функції IsMD5Equal

Основні файли програми застосунку наведені у Додатку А.

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Призначення і умови застосування розробленої програми

Розроблене програмне забезпечення призначене для виконання наступних функцій:

- взаємодію з користувачем через графічний інтерфейс;
- швидкий огляд файлової ієрархії комп'ютера: доступ до дисків, директорій, та файлів різних типів;
- можливість визначення у реальному часі (в процесі введення користувачем) інформації про існування даного файлу (за шляхом), про тип даного файлу та його розмір;
- ведення антивірусної бази: місця збереження інформації про відомі системі віруси;
- можливість взаємодії користувача з антивірусною базою, за допомогою програми: додання, змінення та видалення записів про віруси. Таким чином користувач сам визначає стійкість такого антивіруса: чим більше інформації – тим надійніша його робота та безпечніша робота користувача;
- окрім вибору цільового файлу, вибір цільового вірусу;
- можливість сканування файлів. Сканування здійснюється за двома сценаріями: або перевіряється один обраний файл на наявність одного обраного вірусу, або перевіряється одна обрана директорія на наявність одного вірусу;
- отримання чіткої відповіді на питання, чи є файл небезпечним;
- вивід докладної інформації про процес сканування;
- вивід інформації про зараження для кожного з файлів.

Через те, що програма складається з різних модулів, усі вони мають різний ступінь сумісності з ОС. Сумісність розробленої програми с різними версіями ОС Windows наведена у табл. 4.1.

Таблиця 4.1 – Сумісність програми з версіями ОС Windows

Модулі ОС	WIN32Api (системні функції)	Бібліотеки C++ (нижній рівень програми – антивірус)	Бібліотеки C# (верхній рівень програми – інтерфейс користувача)
Windows 10	Так	Так	Так
Windows 8	Так	Так	Так
Windows 7	Так	Так	Так
Windows XP	Так	Так	Ні
Windows 2000 Professional	Так	Ні	Ні
Windows 98	Так	Ні	Ні
Windows 95	Так	Ні	Ні

Як видно із таблиці, програма може успішно функціонувати лише на останніх версіях ОС Windows, однак для розробки ядра – бібліотеки CPPScanner.dll з антивірусною частиною, можна використовувати і більш ранні версії Windows.

Усі необхідні компоненти для розробки будуть розглянуті нижче.

4.2 Характеристики програми

4.2.1 Бібліотеки та ідентифікатори, що використовувалися у проєкті test.sln

Для роботи у проєкті test.sln (розроблений на C++), необхідному для створення бібліотеки CPPScanner.dll, були використані наступні бібліотеки, впорядковані по мірі важливості:

- <fstream> – для роботи з файловим, потоковим вводом-виводом.

Загалом, необхідна лише на стадії розробки;

- `<iostream>` – для роботи з потоковим вводом-виводом;
- `<string>` – для роботи з рядками, що є об'єктами класу `string`. Для поставлених задач звичайного покажчика на `char` рядок не вистачило;
- `<sstream>` - для роботи з рядками, через інтерфейс потоків. Знадобилася при виводі інформації про покажчики на файли;
- `<openssl/md5.h>` - для кодування даних різними алгоритмами. Звідси використовуємо функцію MD5 для знаходження контрольної суми;
- `"windows.h"` – бібліотека Win32 API для C++.

Також для комфортної роботи, директивою `define` були визначені наступні ідентифікатори:

- `#define ForExport __declspec(dllexport)` – для визначення функції, які будуть доступні для експорту з розробленої dll;
- `#define _CRT_SECURE_NO_WARNINGS` – для використання функції `stricmp` та інших, без зауважень компілятора;
- `#define ConsoleComments` – для виводу коментарів, при тестуванні програми. У разі відсутності необхідності в них, це визначення пропонується закоментувати.

4.2.2 Підключення бібліотеки OpenSSL до проєкту `test.sln`

Як вже було зазначено раніше, бібліотека OpenSSL надає можливості кодування інформації різними алгоритмами [18], у тому числі за допомогою алгоритм `md5` (згідно ТЗ). І якщо зі встановленням та використанням інших бібліотек не виникає труднощів, то з даною бібліотекою є деякі проблеми. Тому, розглянемо процес її встановлення.

Для початку необхідно встановити необхідні компоненти і скомпілювати бібліотеки.

Для того, щоб використати бібліотеки у Visual Studio потрібні:

- архів з вихідними кодами для компіляції бібліотеки версії 1.0.1 з офіціального сайту;
- командний рядок Perl для конфігурації бібліотеки з офіціального сайту;
- середовище розробки Visual Studio, а саме її утиліта «Командний рядок розробника».

Після завантаження файлів, розпаковуємо архів openssl-1.0.1a.tar.gz і його вміст копіюємо в папку C:\openssl. Потім відкриваємо командний рядок Perl від імені адміністратора і пишемо такі команди:

```
cd c:\openssl;
perl Configure VC-WIN32 --prefix = c: \ Temp \ openssl;
ms \ do_ms.
```

Потрібно зауважити, що після виконання другої команди останнім рядком в консолі має бути «Configured for VC-WIN32». Конфігурацію завершено, тепер виконаємо компіляцію. Знаходимо в каталозі Visual Studio «Командний рядок розробника», запускаємо від імені адміністратора і вводимо команди:

```
cd c:\openssl;
nmake -f ms\ntdll.mak;
nmake -f ms\ntdll.mak install.
```

Для використання функцій бібліотеки потрібно підключити до проєкту заголовки, але спочатку їх треба помістити в папку, де встановлене середовище розробки. Для цього переходимо в каталог ..\Visual Studio\VC\include і копіюємо туди папку C: \Temp\openssl\include\openssl.

Також необхідно додати в проєкт файли з розширенням .lib, які були отримані раніше. Додаємо їх у файли ресурсів.

Після цього можна підключати бібліотеку <openssl/md5.h> у звичному форматі.

4.2.3 Бібліотеки та ідентифікатори, що використовувалися у проєкті AV_manager.sln

Для роботи у проєкті AV_manager.sln, необхідному для виконання сканування, створення графічного інтерфейсу та з'єднання з БД, необхідні наступні бібліотеки, підключення яких не є очевидним:

- System.IO – для забезпечення файлового вводу-виводу, та вводу-виводу з пам'яті;
- System.Data.SqlClient – для забезпечення з'єднання з БД та виконання запитів до неї;
- System.Text.RegularExpressions – для написання регулярних виразів та перевірки даних, що додаються у БД на відповідність встановленому шаблону;
- System.Runtime.InteropServices – для взаємодії та керування некерованим кодом, тобто кодом, що виконується поза середовищем виконання (interop services – служби взаємодії). До такого коду і відносять функції бібліотек динамічного компонування. Використання InteropServices дозволяє використовувати атрибут DllImport та клас MarshalAs. Останній вказує, як маршалізувати дані між некерованим кодом та кодом, що керується.

4.2.4 Функції Win32 API, що використовувалися у проєкті AV_manager.sln

Для створення головного алгоритму антивірусу, що виконує порівняння сигнатур файлу та вірусу, та для забезпечення додаткових можливостей системи, було використано функції програмного інтерфейсу Win32. У таблиці 4.2 наведено функції Win32 API, що використовувалися у проєкті AV_manager.sln.

Таблиця 4.2 – Використані функції Win32 API

Застосування методу у програмі	Опис
HANDLE hfile = CreateFileA(filename, FILE_READ_ACCESS, NULL, NULL, OPEN_EXISTING, NULL, NULL)	Функція створює або відкриває каталог, фізичний диск. Функція повертає дескриптор, який може бути використаний для доступу до об'єкта.
DWORD file_size = GetFileSize(hfile, NULL)	Функція видобуває розмір зазначеного файлу. Розмір файлу, який може бути повідомлений за допомогою цієї функції обмежується значенням подвійного слова (DWORD).
HANDLE hMap = CreateFileMappingA(hfile, NULL, PAGE_READONLY, NULL, file_size, NULL)	Функція створює або відкриває іменованний або безіменний об'єкт для заданого файлу.
LPVOID File = MapViewOfFile(hMap, FILE_MAP_READ, NULL, NULL, file_size)	Функція відображає представлення файлу в адресний простір визначеного процесу.
return GetFileAttributesA(filename) != INVALID_FILE_ATTRIBUTES	Функція повертає встановлені атрибути в стилі файлової системи FAT визначеного файлу або каталогу. В даному випадку перевіряємо, чи існує файл.
CloseHandle(hfile)	Функція закриває дескриптор відкритого об'єкта.

4.2.5 Опис розроблених функцій бібліотеки CPPScanner.dll

Для компіляції функцій бібліотеки CPPScanner.dll, були розроблені спеціальні функції. Вони визначають головний смисл, однак не мають зв'язування у стилі мови С, та не призначені для експорту. Розглянемо їх призначення:

– `char* FindMD5(LPVOID File, DWORD offset = 0, DWORD length = 10)`. Функція приймає наступні параметри: покажчик на початок файлу в адресному просторі, встановлене зміщення, встановлений розмір послідовності. Функція визначає інформацію, збережену у файлі, у вигляді потоку байт. Це звичайно відбувається за заданим зміщенням покажчика та для заданої довжини байт. Далі, ця інформація кодується функцією MD5 бібліотеки openssl. Після кодування у 16-байтовий вид, код переводиться у 32 символний рядок, який і повертається у якості кінцевого значення;

– `bool IsMD5Equal(LPVOID File, AVRecord avr)`. Дана функція визначає основне порівняння файлу з вірусом. IsMD5Equal проводить дії, які аналогічні діям функції FindMD5. Однак після отримання контрольної суми файлу, вона перевіряє, чи співпадає хеш файлу з хешем вірусу. У разі підтвердження, функція повертає значення істина. Це фактично може свідчити, що файл є небезпечним і визначений як вірус. У разі, якщо хеші не співпадають – файл «здоровий»;

– `ScannerResult ScannerAlgorithm(PCSTR filename, AVRecord avr, PSTR& handle1, PSTR& handle2, PSTR &FileStartMem, PCSTR& FileHash)`. Функція виконує головний алгоритм перевірки. Вона приймає значення імені файлу, запис про вірус та ряд параметрів, переданих за посиланням. Такі параметри в даному контексті називаються додатковими. Вони надають необов'язкову інформацію користувачу або розробнику про хід виконання алгоритму. Функція поступово виконує методи Win32 API по роботі з файлами: CreateFileA, GetFileSize, CreateFileMappingA, MapViewOfFile. Після цього, вона отримує покажчик на початок файлу в адресному просторі процесу. Далі

виконуються перевірки коректності даних та заповнюється і виводиться на консоль додаткова інформація. Далі, отримані параметри передаються функції FindMD5, яка і перевіряє, чи є файл зараженим. Після перевірки відбувається повернення результату виконання функції, значення переліку ScannerResult;

– ScannerResult ScannerAlgorithmSimple(PCSTR filename, AVRecord avr).

Функція головного алгоритму програми, однак спрощена: вона не виводить та не працює з додатковими параметрами.

Тепер опишемо функції, що мають зв'язування у стилі мови C, та призначені для експорту та використання у проєкті AV_Manager:

– orExport bool isFileExist(PCSTR filename). Функція перевірки, чи існує файл за заданим шляхом;

– ForExport unsigned long FileSize(PCSTR filename). Функція знаходить розмір файлу за заданим шляхом;

– ForExport char* FindMD5ForTest(const char * str_for_md5, int n). Функція знаходить хеш для рядку символів заданої довжини;

– ForExport ScannerResult isFileInfected(PCSTR filename, const char* name, DWORD offset, DWORD length, const char* hash, PSTR &handle1, PSTR& handle2, PSTR &FileStartMem, PCSTR& FileHash). Функція перевіряє, чи є файл небезпечним, з урахуванням зазначеного вірусу. Тобто йде перевірка 1-1. Функція застосовує ScannerAlgorithm;

– ForExport ScannerResult isFileInfectedSimple(PCSTR filename, const char* name, DWORD offset, DWORD length, const char* hash, PSTR& handle1, PSTR& handle2, PSTR& FileStartMem, PCSTR& FileHash). Функція аналогічна функції isFileInfected, однак спрощена, тому і застосовує спрощену версію алгоритму порівняння з вірусом ScannerAlgorithmSimple.

4.3 Вхідні та вихідні дані

Інформація щодо вхідних та вихідних даних була наведена у п. 1.4.3. Розглянемо типи даних, що використовуються у Win API та структуру БД.

Бібліотека «windows.h» додатково, забезпечує множину типів даних. Для безпомилкової роботи програми та для більш точного розуміння предметної області в розробленому програмному забезпеченні використовувалися наступні типи даних:

- основні: BOOL, BYTE, DWORD, NULL;
- дескрипторні: HANDLE. Дескриптор - це ідентифікатор будь-якого об'єкта. Для різних типів об'єктів існують різні дескриптори. В даному проєкті було застосовано дескриптор файлу;
- строкові: LPSTR, PCSTR, TCHAR;
- додаткові: LPVOID. Ці типи даних використовуються в деяких функціях.

Структура БД складається з однієї таблиці, а її записи повністю повторюють відповідні структури у проєктах написаних мовою C++ – AVRecord та AVSignature, і мовою C# – AntiVirusRecord.

БД була створена безпосередньо за допомогою вбудованого інтерфейсу SQL Server у MVS. Робота з такою базою через посередництво MVS зображена на рис. 4.1.

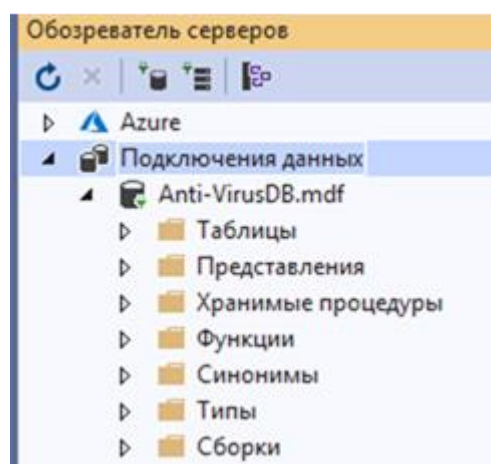


Рисунок 4.1 – Вікно взаємодії з БД

Код створення таблиці AVList наведено у Додатку А.

5 КЕРІВНИЦТВО ОПЕРАТОРА

5.1 Призначення програми

Розроблене програмне забезпечення призначене для знешкодження вірусів і різного роду шкідливого ПЗ, з метою збереження інформації користувача на ПК.

5.2 Умови використання програми

5.2.1 Вимоги до оператора

В цілому, програма може підійти як звичайному користувачу, так і учню для ознайомлення з роботою простого сканеру. До оператора висуваються наступні вимоги:

- розуміння предметної області. Сюди відносимо розуміння логіки роботи сканеру, для аналізу ще й додаткових параметрів. Якщо ж програма використовується лише для отримання відповіді, чи є файл небезпечним, то розуміння предметної області взагалі не обов'язкове;

- навички роботи з мишею, клавіатурою;
- навички роботи з графічним інтерфейсом;
- володіння інформацією про призначення основних елементів керування більшості програм;

- знання з організації файлової системи Windows;

- знання про елементарну організацію антивірусної бази.

5.2.2 Вимоги до технічних засобів та операційної системи

Робота застосунку має відбуватись під управлінням операційної системи Windows 10. Це обумовлює склад та системні вимоги до технічних засобів (див. п. 1.4.4).

5.3 Виконання програми

5.3.1 Запуск програми

Перед запуском програми потрібно попередньо упевнитися у наявності всіх необхідних бібліотек для роботи програми (у першу чергу це CPPScanner.dll) та в наявності бази даних Anti-VirusDB.mdf.

Звертання до програми передбачене одним способом – через файл, що виконується. Звертання до програми через командний рядок для користувача не передбачене. Після запуску файлу AV_Manager.exe з'являється головна форма програми, наведена на рис. 5.1.

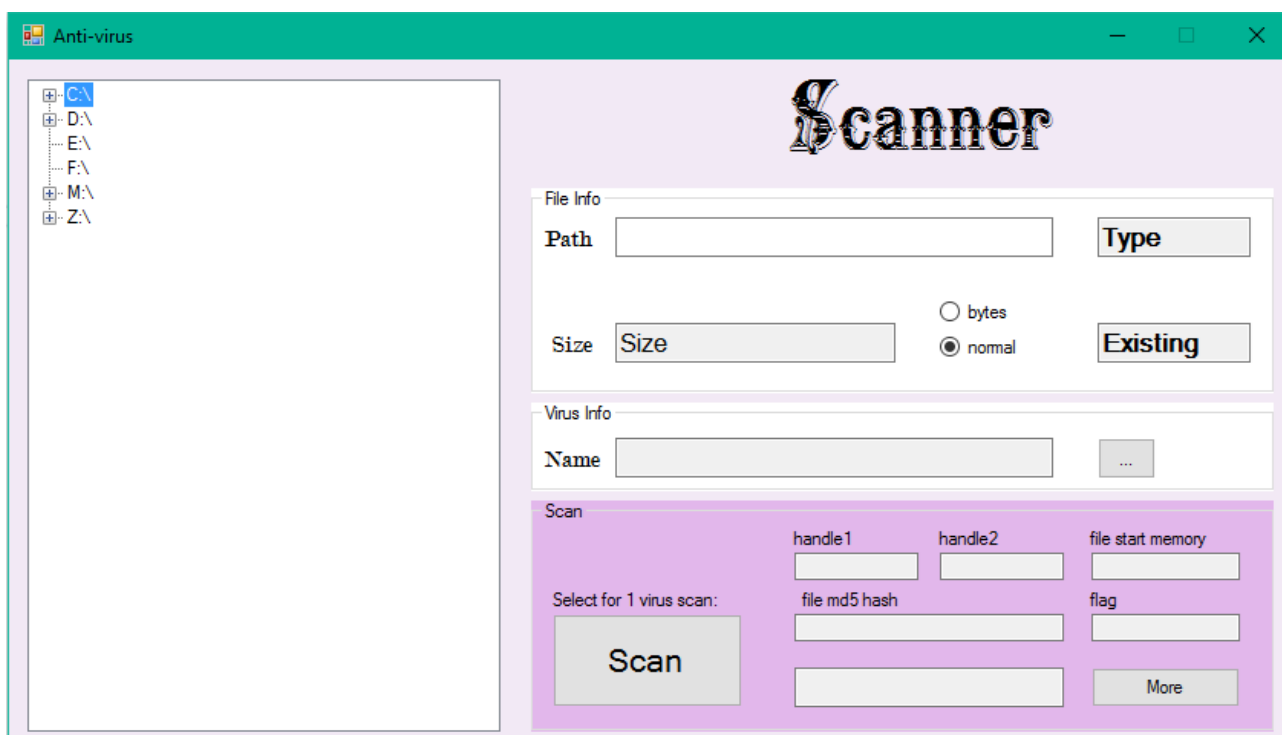


Рисунок 5.1 – Головна форма програми

Інтерфейс головної форми розділений на чотири частини, які відповідають за окремі фази сканування:

– за допомогою FileTree (у лівій частині рис. 5.1) оператор може оглянути свою файлову систему у вигляді дерева. Для розкриття та перегляду вмісту директорії потрібно натиснути на відповідний вузол. Для того, щоб

обрати конкретний файл, або каталог, необхідно двічі натиснути на ньому. Після цього його повне ім'я переміститься до області вікна «File Info»;

- File Info, це область, що надає користувачу інформацію про поточний файл, шлях до якого зберігається у текстовому полі. Його можна редагувати як вручну, так і вибравши відповідний пункт з дерева FileTree. У режимі реального часу користувачу надається інформація про параметри файлу: чи існує він, який в нього тип (директорія або простий файл), який в нього розмір. Поля з параметрами не можна редагувати, однак для поля з розміром файлу можна обрати формат, натиснувши на перемикач;

- Virus Info, це область, що дозволяє взаємодіяти користувачеві з антивірусною базою. У текстовому полі, він може бачити поточний вибраний антивірус для перевірки, а безпосередньо здійснити вибір він може, натиснувши кнопку з 3 точками, що надасть доступ для перегляду та взаємодії з БД;

- Scan, це поле для проведення сканування, та перегляду його результатів. У полях з додатковою інформацією наведено значення дескрипторів, адреса початку файлу, хеш файлу, знайдений за заданим зміщенням та довжиною та флаг, з яким завершується сканування. Результат сканування відображається у спеціально відведеному для цього полі.

5.3.2 Робота з антивірусною базою. Сканування.

Натиснувши на кнопку з трьома крапками головної форми (рис.5.1), користувач може редагувати вміст антивірусної бази. Форма для редагування БД наведена на рис. 5.2.

Елементи форми наступні:

- кнопка «Select» призначена для переносу обраного зі списку вірусу на головну форму, та позначення його як цільового;
- кнопка «Cancel» закриває форму;

- кнопка «Edit» відкриває форму редагування запису про вірус у відповідному вікні з параметрами вибраного зі списку вірусом;
- кнопка «Add new» відкриває форму додавання запису про вірус;
- кнопка «Delete» видаляє із системи визначений користувачем вірус.

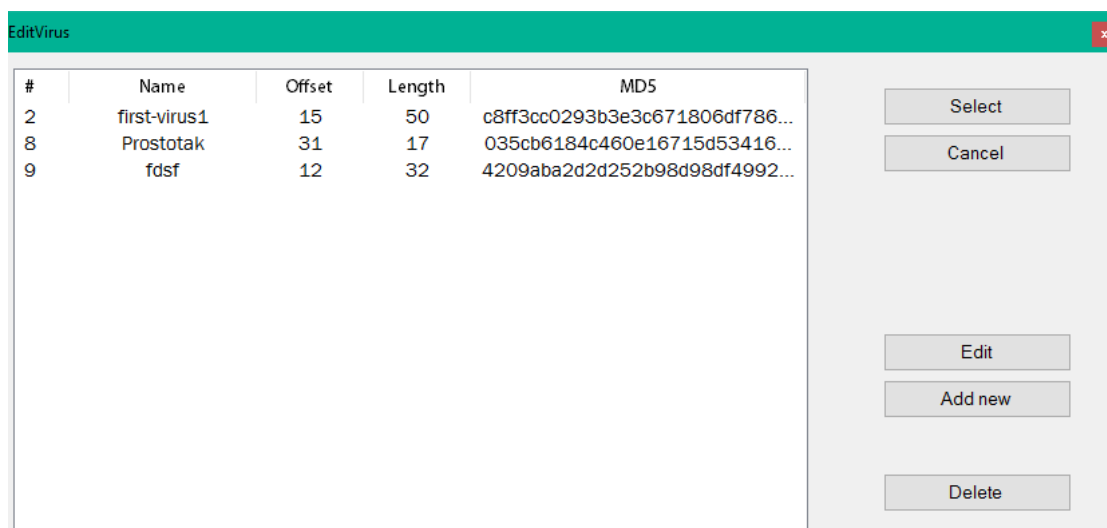


Рисунок 5.2 – Форма редагування антивірусної бази

На рис. 5.3 зображена форма з редагуванням та додаванням запису про вірус до БД.

The screenshot shows a window titled 'Edit Virus Param' with input fields for 'Virus name', 'Offset', 'Length', and 'MD5 Hash'. The 'Virus name' field contains 'fdsf'. The 'Offset' field contains '12', the 'Length' field contains '32', and the 'MD5 Hash' field contains '4209aba2d2d252b98d98df499206b6d1'. There is an 'Affirm' button at the bottom.

Рисунок 5.3 – Форма редагування запису про вірус антивірусної бази

Сканування проводиться для обраного у області «File Info» файлу (або директорії) та для визначеного у області «Virus Info» вірусу. Сканування відбувається після натискання кнопки «Scan». Приклад результатів сканування для одного файлу зображено на рис. 5.4.

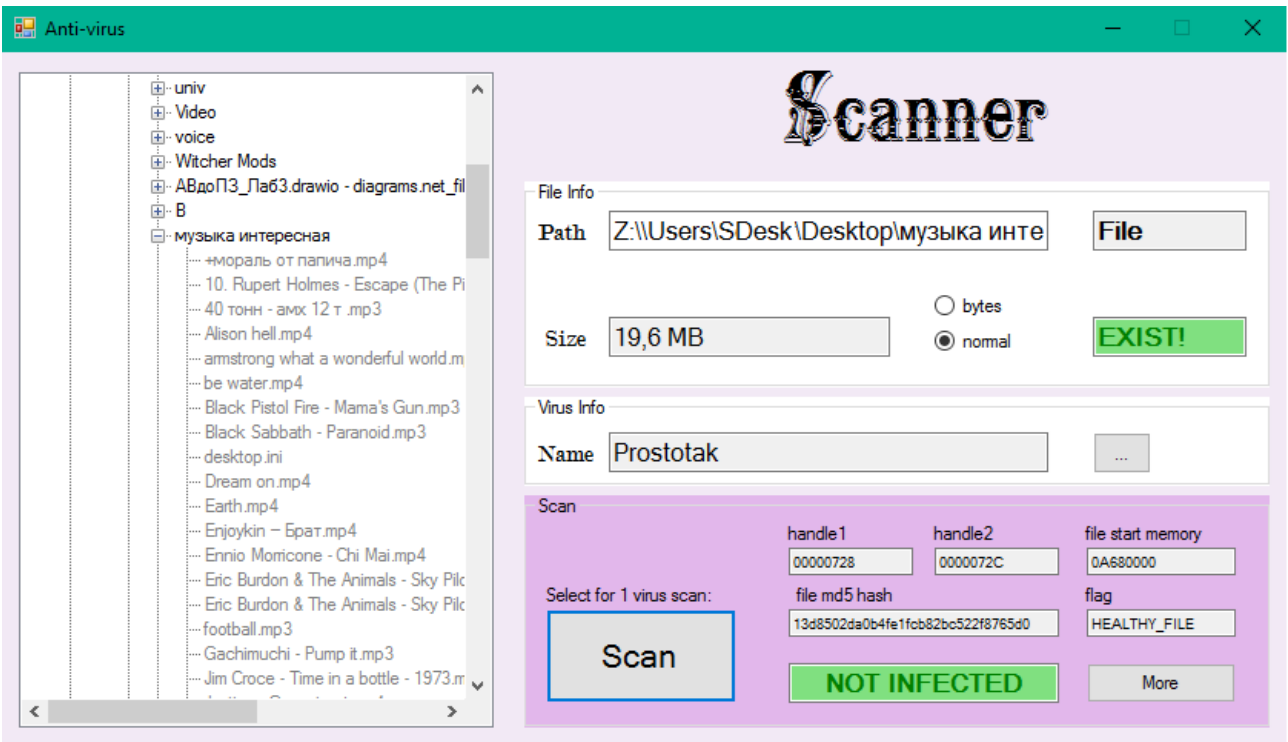


Рисунок 5.4 – Результаты сканування

Детальну інформацію про результати сканування директорії можна побачити, натиснувши на кнопку More (рис. 5.4). Форма з детальною інформацією зображена на рис. 5.5.

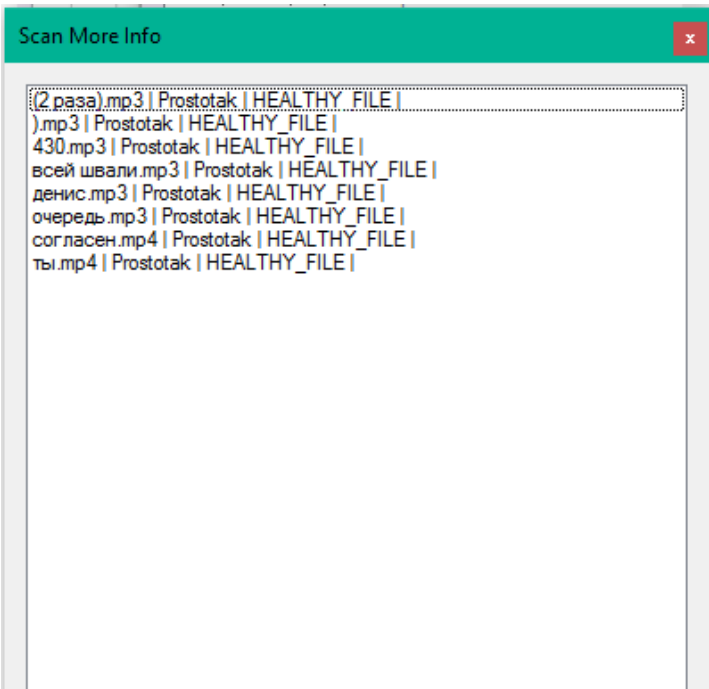


Рисунок 5.5 – Результаты сканування директорії

5.4 Повідомлення оператору

Оператор отримує повідомлення у вигляді вікон. Можливі повідомлення оператору наведені у табл. 5.1.

Таблиця 5.1 – Повідомлення оператору

Повідомлення	Причина	Подальші дії оператора
1	2	3
Некоректні записи вірусів у БД.	Повідомлення виникає, якщо у БД зберігаються записи, формат яких не відповідає формату для відповідного запису вірусу.	Оператор має проаналізувати число некоректних записів, закрити вікно і продовжити роботу з відкритою формою редагування БД. Більше це вікно не з'явиться. Для усунення цієї помилки, необхідно вручну змінити записи про віруси у БД на коректні.
Не обрано вірусу для перевірки при скануванні.	Спроба зробити сканування для вірусу, що не був обраний з БД.	Оператор має обрати вірус, відкривши вікно роботи з антивірусною БД. Прапорець успішного вибору вірусу - поява його імені у відповідному текстовому полі на головній формі.

Продовження таблиці 3.1

1	2	3
Не обрано файлу або директорії для сканування.	Спроба зробити сканування неіснуючої, або невибраної директорії.	Оператор має або вибрати файл чи директорію, або вказати той файл чи директорію, які реально існують на комп'ютері.
Некоректний формат даних вірусу.	Повідомлення виникає у разі введення некоректних даних для вірусу у випадку редагування чи додавання запису у БД.	Оператор має задавати коректні дані, якщо хоче додати чи відредагувати запис у антивірусній базі.
Відсутність доступу до файлів	Це повідомлення виникає у випадку, коли програма не має доступу до читання деяких файлів комп'ютера, або вони виконуються у поточний час (наприклад, системні файли).	Оператор або не може взагалі сканувати цей проблемний файл чи директорію в цілому, або має закрити файли що виконуються чи відкриті в системі у поточний момент.
Неможливо відкрити файл CPPScanner.dll.	Повідомлення виникає, якщо у відповідній директорії відсутня бібліотека CPPScanner.dll.	Оператор має скопіювати файл CPPScanner.dll у папку, де знаходиться exe-файл з менеджером: AV_Manage.exe.
Неможливо відкрити антивірусну базу	Повідомлення виникає, якщо у відповідній директорії відсутній файл БД Anti-VirusDB.mdf.	Оператор має скопіювати файл Anti-VirusDB.mdf у папку, де знаходиться файл AV_Manage.exe.

Приклад повідомлення «Некоректні записи вірусів у БД.» наведений на рис. 5.6.

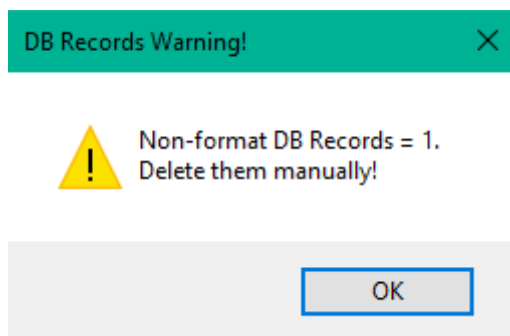


Рисунок 5.6 – Помилка читання записів з БД

ВИСНОВКИ

Під час виконання дипломної кваліфікаційної роботи бакалавра було розроблено програмне забезпечення, що виконує функції антивірусної програми.

Було проведено огляд типів та характеристик вірусів, а також існуючих антивірусних програм. У результаті такого аналізу було прийнято рішення розробляти антивірусну програму у вигляді сканеру сигнатур.

Для програми було реалізовано інструментарій для роботи з антивірусною базою, що дозволяє користувачу легко її заповнювати, підвищуючи рівень безпеки системи.

Було розроблено алгоритми сканування, з використанням функцій, що безпосередньо взаємодіють з операційною системою Windows. Розроблені методи було зібрано у вигляді бібліотеки динамічного компонування, створеної мовою C++.

Мовою C# було поєднано усі компоненти системи та розроблено максимально простий та зрозумілий інтерфейс, що ґрунтується на поняттях: файл, вірус, сканування.

Одною з ключових переваг розробленого продукту є закладення у проєкт основи для його подальшого розвитку та модернізації. Зважаючи на прозорість компонентів, модульність та простоту алгоритмів, цей програмний продукт має наступні перспективи до подальшої розробки та покращення:

- створення алгоритму, який повністю заснований на функціях системи Win32 API;

- більш детальне, інформаційне та повне сканування усієї системи, покращення надійності її функціонування.

Антивірусна програма призначена для використання під управлінням операційної системи Windows 10.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. История развития вирусов [Электрон. ресурс]. – Режим доступа: <https://2hpc.ru/kaspersky-antivirus-kav/>.
2. Компьютерные вирусы. История создания, классификация, способы и признаки заражения [Электрон. ресурс]. – Режим доступа: <https://sites.google.com/site/antivirusnyeprogrammyivirusy/home/komputernye>.
3. Создаём PE-вирус №1 [Электрон. ресурс]. – Режим доступа: <https://xakep.ru/2007/04/23/37880/#toc02>.
4. Лаборатория Касперского [Электрон. ресурс]. – Режим доступа: <https://www.kaspersky.ru/home-security/v3>.
5. Avast [Электрон. ресурс]. – Режим доступа: <https://www.avast.ua/download-thank-you.php?product=FAV-PPC&locale=uk-ua>.
6. AVG Technologies [Электрон. ресурс]. – Режим доступа: <https://www.avg.com/ru-ru/download-thank-you.php?product=FREEGSR>.
7. Антивирус Касперского 2017 – плюсы и минусы [Электрон. ресурс]. – Режим доступа: <https://2hpc.ru/kaspersky-antivirus-kav/>.
8. Проблемы защиты информации в компьютерной сети [Электрон. ресурс]. – Режим доступа: <https://moluch.ru/archive/115/31145/>.
9. Основы работы с Win API [Электрон. ресурс]. – Режим доступа: <http://www.delphikingdom.com/asp/viewitem.asp?catalogid=169#03>.
10. Cannot Debug Unmanaged Dll from C# [Electronic resource]. – Access mode: <https://stackoverflow.com/questions/1744888/cannot-debug-unmanaged-dll-from-c-sharp>.
11. Passing strings between managed and unmanaged code [Electronic resource]. – Access mode: <https://www.codeproject.com>.
12. Loading c++ dll function having parameter (char *) into c# [Electronic resource]. – Access mode: <https://www.codeproject.com>.
13. Returning Strings from a C++ API to C# [Electronic resource]. – Access mode: <https://limbioliong.wordpress.com/2011/06/16/returning-strings-from-a-c-api/>.

14. Что такое extern «C»? [Электрон. ресурс]. – Режим доступа: <https://alex-siew.livejournal.com/2489.html>.

15. C# DllImport with C++ boolean function [Electronic resource]. – Access mode: <https://stackoverflow.com/questions/4608876/c-sharp-dllimport-with-c-boolean-function-not-returning-correctly>.

16. Адресное пространство процесса в Windows [Электрон. ресурс]. – Режим доступа: <http://datadump.ru/process-address-space/>.

17. Virus Share [Electronic resource]. – Access mode: <https://virusshare.com/>.

18. «OpenSSL docs» [Electronic resource]. – Access mode: <https://docs.huihoo.com/doxygen/openssl/1.0.1c/annotated.html>.

ДОДАТОК А
Текст програми

A.1 Код бази даних

```
CREATE TABLE [dbo].[AVList] (
    [Id]      INT          IDENTITY (1, 1) NOT NULL,
    [Name]    TEXT         NOT NULL,
    [Offset]  INT          NOT NULL,
    [Length]  INT          NOT NULL,
    [MD5]     CHAR (32) NOT NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC),
    CONSTRAINT [UC_MD5] UNIQUE NONCLUSTERED ([Id] ASC, [MD5] ASC)
);
```

A.2 Код файлу test.sln

```
#define ForExport __declspec(dllexport)
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <string>
#define ConsoleComments
#include "windows.h";
#include <fstream>;
#include <openssl/md5.h>;
#include <sstream>
using namespace std;
// для вывода комментариев в консоль
class Comments {
public:
    static void ConsoleOut(string message1, PCSTR message2 = "") {
#ifdef ConsoleComments;
        cout << "\n" << "-----" << "\n";
        cout << message1 << message2;
        cout << "\n" << "-----" << "\n";
#endif
    }
};
// результат проверки файл на вирус
```

```

enum ScannerResult
{
    INVALID_HANDLE,
    FILE_NULL,
    SMALL_FILE,
    INFECTED_FILE,
    HEALTHY_FILE
};

// класс сигнатуры вируса: смещение, длина, хеш
class AVSignature {
public:
    // 32-битное беззнаковое целое
    DWORD offset; // смещение последовательности
    char hash[33]; // хэш
    DWORD length; // размер последовательности
    AVSignature() {
        this->offset = 0;
        this->length = 0;
        // заполняем нулями блок памяти hash длиной в 16 байт (4*DWORD)
        //memset(this->hash, 0, sizeof(this->hash));
    }
    AVSignature(DWORD offset, DWORD length, const char* hash) {
        this->offset = offset;
        this->length = length;
        strcpy(this->hash, hash);
    }
    void ConsoleOutput()
    {
        cout << "Offset = " << offset << "\n";
        cout << "Length = " << length << "\n";
        cout << "Hash = " << string(hash) << "\n";
    }
};

// класс записи о зловреде в АББ (в антивирусной базе)
class AVRecord {
public:
    string name; // имя вируса

```

```

AVSignature signature; // сигнатура вируса
AVRecord()
{
    this->name = "";
    this->signature = AVSignature();
}
AVRecord(string name, DWORD offset, DWORD length, const char* hash)
{
    this->name = name;
    this->signature = AVSignature(offset, length, hash);
}
void ConsoleOutput()
{
    cout << "Virus name:" << this->name << "\n";
    signature.ConsoleOutput();
}
};
// Главная часть алгоритма антивируса.
bool IsMD5Equal(LPVOID File, AVRecord avr)
{
    unsigned char md5digest[MD5_DIGEST_LENGTH];
    MD5((PBYTE)((DWORD)File + avr.signature.offset),
avr.signature.length, md5digest);
    char* mdString = new char[33];
    for (int i = 0; i < MD5_DIGEST_LENGTH; i++)
        sprintf(&mdString[i * 2], "%02x", (unsigned int)md5digest[i]);
    if (strcmp(mdString, avr.signature.hash) == 0) return true;
    return false;
}
// Для вывода хеша файла на экран. Так же задается смещение и длина
char* FindMD5(LPVOID File, DWORD offset = 0, DWORD length = 10)
{
    unsigned char md5digest[MD5_DIGEST_LENGTH];
    MD5((PBYTE)((DWORD)File + offset), length, md5digest);
    char* mdString = new char[33];
    for (int i = 0; i < MD5_DIGEST_LENGTH; i++)
        sprintf(&mdString[i * 2], "%02x", (unsigned int)md5digest[i]);

```

```

        return mdString;
    }
    // Алгоритм антивируса. Проверяем файл на наличие вируса. Если файл
заражен - true, иначе - false
    ScannerResult ScannerAlgorithm(PCSTR filename, AVRecord avr,
        PSTR& handle1, PSTR& handle2, PSTR &FileStartMem, PCSTR& FileHash)
{
    //---ПОЛУЧЕНИЕ ПРОЕКЦИИ ФАЙЛА В АДРЕСНОЕ ПРОСТРАНСТВО ВЫЗЫВАЮЩЕГО
ПРОЦЕССА-----
    // вывод инф. о проверяемом файле
    Comments::ConsoleOut("Try to check file - ", filename);
    // открываем уже созданный файл с доступом к чтению. Функция
возвращает дескриптор, который может быть использован для доступа к объекту
    HANDLE hfile = CreateFileA(filename, FILE_READ_ACCESS, NULL, NULL,
OPEN_EXISTING, NULL, NULL);
    // если произошла ошибка открытия файла
    if (hfile == INVALID_HANDLE_VALUE) {
        Comments::ConsoleOut("Invalid file-handle value error");
        // уничтожаем дескриптор
        CloseHandle(hfile);
        return ScannerResult::INVALID_HANDLE;
    }
    // находим размер файла в байтах
    DWORD file_size = GetFileSize(hfile, NULL);
    // функция открывает именованный объект отображенного в памяти
(проецируемого) файла для заданного файла.
    HANDLE hMap = CreateFileMappingA(hfile, NULL, PAGE_READONLY, NULL,
file_size, NULL);
    if (hMap == INVALID_HANDLE_VALUE) {
        Comments::ConsoleOut("Invalid file-mapping-handle value
error");
        CloseHandle(hMap);
        return ScannerResult::INVALID_HANDLE;
    }
    // функция отображает представление проецируемого файла в адресное
пространство вызывающего процесса.

```

```

LPVOID File = MapViewOfFile(hMap, FILE_MAP_READ, NULL, NULL,
file_size);

    if (File == NULL) {
        Comments::ConsoleOut("LPVOID File is NULL");
        CloseHandle(hfile);
        CloseHandle(hMap);
        return ScannerResult::FILE_NULL;
    }
    //-----ПОИСК КОНТРОЛЬНОЙ СУММЫ И СРАВНЕНИЯ ЕЁ С ХЕШЕМ ВИРУСА--
    // Если файл слишком маленький, то пропускам запись
    if (file_size < avr.signature.offset + avr.signature.length) {
        Comments::ConsoleOut("File is too small");
        return ScannerResult::SMALL_FILE; // иначе мы просто не сможем
пройти по смещению и будет ошибка
    }
    //-----ВСПОМОГАТЕЛЬНАЯ ИНФОРМАЦИЯ В КОНСОЛЬ-----
    cout << "\nFile size is " << file_size << " bytes\n";
    // адрес файла в 16-ричной системе. Разумеется может быть разным,
    тк относителен.
    cout << "Memory address of file hex: " << File << "\n";
    // адрес файла в 10-ричной системе.
    cout << "Memory address of file dec: " << (DWORD)File << "\n";
    // адрес файла + смещение в 10-ричной системе.
    cout << "Memory address of file + offset dec: " << (DWORD)File +
avr.signature.offset << "\n\n";
    // выводим хеш вируса
    cout << "\nVirus hash:\n" << string(avr.signature.hash);
    // выводим хеш файла по смещению с заданой длиной
    cout << "\nFile hash:\n" << string(FindMD5(File,
avr.signature.offset, avr.signature.length));
    //---ВСПОМОГАТЕЛЬНАЯ ИНФОРМАЦИЯ ДЛЯ ВОЗВРАТА ИЗ ФУНКЦИИ-----
    // переводим из указателя с адресом в строку типа string
    ostringstream oss;
    oss << hfile;
    string fin1 = oss.str();
    oss.str("");
    oss.clear();

```

```

    oss << hMap;
    string fin2 = oss.str();
        oss.str("");
        oss.clear();
    oss << File;
    string fin3 = oss.str();
        oss.str("");
        oss.clear();

    // переводим из строки типа string в char*
    strcpy(handle1, fin1.c_str());
    strcpy(handle2, fin2.c_str());
    strcpy(FileStartMem, fin3.c_str());
    FileHash = (const char*)FindMD5(File, avr.signature.offset,
avr.signature.length);
    printf("\n");
    CloseHandle(hfile);
    CloseHandle(hMap);

    //-----ПРОВЕРКА ФАЙЛА НА ВИРУС-----
    // возвращаем результат сравнения хеша сигнатуры файла и вируса
    if (IsMD5Equal(File, avr)) return ScannerResult::INFECTED_FILE;
    else return ScannerResult::HEALTHY_FILE;
}

// Упрощенный вариант алгоритма антивируса. Без дополнительной
информации

ScannerResult ScanerAlgorithmSimple(PCSTR filename, AVRecord avr) {
    Comments::ConsoleOut("Try to check file - ", filename);
    HANDLE hfile = CreateFileA(filename, FILE_READ_ACCESS, NULL, NULL,
OPEN_EXISTING, NULL, NULL);
    if (hfile == INVALID_HANDLE_VALUE) {
        Comments::ConsoleOut("Invalid file-handle value error");
        // уничтожаем дескриптор
        CloseHandle(hfile);
        return ScannerResult::INVALID_HANDLE;
    }
    DWORD file_size = GetFileSize(hfile, NULL);
    HANDLE hMap = CreateFileMappingA(hfile, NULL, PAGE_READONLY, NULL,
file_size, NULL);

```

```

        if (hMap == INVALID_HANDLE_VALUE) {
            Comments::ConsoleOut("Invalid file-mapping-handle value
error");

            CloseHandle(hMap);
            return ScannerResult::INVALID_HANDLE;
        }
        LPVOID File = MapViewOfFile(hMap, FILE_MAP_READ, NULL, NULL,
file_size);
        if (File == NULL) {
            Comments::ConsoleOut("LPVOID File is NULL");
            CloseHandle(hfile);
            CloseHandle(hMap);
            return ScannerResult::FILE_NULL;
        }
        if (file_size < avr.signature.offset + avr.signature.length) {
            Comments::ConsoleOut("File is too small");
            return ScannerResult::SMALL_FILE; // иначе мы просто не сможем
пройти по смещению и будет ошибка
        }
        CloseHandle(hfile);
        CloseHandle(hMap);
        if (IsMD5Equal(File, avr)) return ScannerResult::INFECTED_FILE;
        else return ScannerResult::HEALTHY_FILE;
    }
    extern "C" {
        // проверяет, существует ли указанный файл или каталог
        ForExport bool isFileExist(PCSTR filename)
        {
            Comments::ConsoleOut("Is file exist - ", filename);
            // если функция завершается ошибкой, возвратится значение
INVALID_FILE_ATTRIBUTES, с ним и сравниваем
            // INVALID_FILE_ATTRIBUTES эквивалентно 0xffffffff -
максимальному значению переменной
            return GetFileAttributesA(filename) !=
INVALID_FILE_ATTRIBUTES;
        }
    }

```

```

// определяет размер файла, диска и пр. по пути
ForExport unsigned long FileSize(PCSTR filename) {
    HANDLE hfile = CreateFileA(filename, FILE_READ_ACCESS, NULL,
    NULL, OPEN_EXISTING, NULL, NULL);
    // если произошла ошибка открытия файла
    if (hfile == INVALID_HANDLE_VALUE) {
        // уничтожаем дескриптор
        CloseHandle(hfile);
        return 0;
    }
    // находим размер файла в байтах
    DWORD file_size = GetFileSize(hfile, NULL);
    CloseHandle(hfile);
    return file_size;
}
// функция возвращает 32 символьную строку представления хеша в 16-
ричном формате. Хеш берем для заданной строки для n первых символов
ForExport char* FindMD5ForTest(const char * str_for_md5, int n)
{
    // 16 байт под хеш
    unsigned char md5digest[MD5_DIGEST_LENGTH];
    // находим хеш для n символов строки и помещаем его в
md5digest

    MD5((const unsigned char*)str_for_md5, n, md5digest);
    // выводим хеш в виде 32 символьной строки в 16-ричном формате
    /*cout << "\nMD5digest:\n";
    for (int i = 0; i < MD5_DIGEST_LENGTH; i++) {
        printf("%02x", md5digest[i]);
    };
    printf("\n");*/
    // помещаем строчное представление хеша и возвращаем из
функции

    char* mdString = new char[33];
    for (int i = 0; i < MD5_DIGEST_LENGTH; i++)
        sprintf(&mdString[i * 2], "%02x", (unsigned
int)md5digest[i]);

```

```

        return mdString;
    }
    // функция возвращает результат проверки файла на наличие вируса:
    bool; принимает параметры записи о вирусе и формирует соответствующий объект
    ForExport ScannerResult isFileInfected(PCSTR filename, const char*
name, DWORD offset, DWORD length, const char* hash,
        PSTR &handle1, PSTR& handle2, PSTR &FileStartMem, PCSTR&
FileHash)
    {
        // инициализируем объект записи о зловреде
        AVRecord avr = AVRecord(string(name), offset, length, hash);
        // выводим информацию про проверяемый вирус
        avr.ConsoleOutput();
        // возвращает результат проверки
        return ScannerAlgorithm(filename, avr, handle1, handle2,
FileStartMem, FileHash);
    }
    // функция возвращает результат проверки файла на наличие вируса.
Упрощенный вариант
    ForExport ScannerResult isFileInfectedSimple(PCSTR filename, const
char* name, DWORD offset, DWORD length, const char* hash,
        PSTR& handle1, PSTR& handle2, PSTR& FileStartMem, PCSTR&
FileHash)
    {
        // инициализируем объект записи о зловреде
        AVRecord avr = AVRecord(string(name), offset, length, hash);
        // выводим информацию про проверяемый вирус
        avr.ConsoleOutput();
        // возвращает результат проверки
        return ScannerAlgorithmSimple(filename, avr);
    }
}

```

A.3 Код файлу AV_Manager.sln

```

AV_Manager.sln
ScannerDLL

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Runtime.InteropServices;
using System.IO;
namespace AV_manager
{
    public class ScannerDLL
    {
        //        для        теста        и        изменения        библиотеки
        @"Z:\Users\Sergey\Desktop\univ\3 семестр универ\Курсач\test\Debug\test.dll";
        // для соединения с длл с++
        private const string CppFunctionDll = @"CPPScanner.dll";
        // флаги окончания поиска вируса
        [Flags]
        public enum ScannerResult : uint
        {
            INVALID_HANDLE, // дескриптор файла не найден
            FILE_NULL, // адресс файла не найден
            SMALL_FILE, //размер файла недостаточен для проверки
            INFECTED_FILE, // файл заражен
            HEALTHY_FILE // файл без вирусов
        }
        // существует ли файл
        [DllImport(CppFunctionDll, CallingConvention =
CallingConvention.Cdecl)]
        [return: MarshalAs(UnmanagedType.I1)]
        public static extern bool isFileExist(string filename);
        // найти мд5 для заданой строки длиной n
        [DllImport(CppFunctionDll, CallingConvention =
CallingConvention.Cdecl)]
        public static extern IntPtr FindMD5ForTest(string str_for_md5,
int n);

        // заражен ли файл + доп. информация по ходу

```

```

        [DllImport(CppFunctionDll, CallingConvention =
CallingConvention.Cdecl)]
        public static extern ScannerResult isFileInfected(string
filename, string name, uint offset, uint length, string hash,
        out IntPtr handle1, out IntPtr handle2, out IntPtr
FileStartMem, out IntPtr FileHash);
        // заражен ли файл + without доп. информация по ходу
        [DllImport(CppFunctionDll, CallingConvention =
CallingConvention.Cdecl)]
        public static extern ScannerResult isFileInfectedSimple(string
filename, string name, uint offset, uint length, string hash);
        // определяет размер файла, диска, папки и пр.
        [DllImport(CppFunctionDll, CallingConvention =
CallingConvention.Cdecl)]
        public static extern ulong FileSize(string filename);
    }
}

```

A.4 Код файлу AntiVirusRecord

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Text.RegularExpressions;
namespace AV_manager
{
    public class AntiVirusRecord
    {
        private uint id;
        public string name;
        public uint offset, length;
        public string md5;
        public AntiVirusRecord()
        {

```

```

        SetAllNull();
    }
    public AntiVirusRecord(uint i, string n, uint o, uint l, string m)
    {
        SetAllNull();
        // имя вируса не больше 60 символов
        if (n.Length < 60)
        {
            // мд5 32 символа и все символы 16-ричной системы
            if (m.Length == 32 && Regex.IsMatch(m, @"^[a-fA-F0-9]+$"))
            {
                id = i;
                name = n;
                offset = o;
                length = l;
                md5 = m;
            }
        }
    }
    private void SetAllNull()
    {
        id = 0;
        name = "None";
        offset = length = 0;
        md5 = "";
    }
    public bool isNull()
    {
        return id == 0;
    }
    public uint GetID()
    {
        return id;
    }
    public string GetString()
    {

```

```
        return "Virus Name: " + name + "; offset = " + offset + ";  
length = " + length + "; md5: " + md5 + "\n";  
    }  
    public string GetName()  
    {  
        return name;  
    }  
    public uint GetOffset()  
    {  
        return offset;  
    }  
    public uint GetLength()  
    {  
        return length;  
    }  
    public string GetMD5()  
    {  
        return md5;  
    }  
}  
}
```

ДОДАТОК Б
Копії слайдів презентації

ДИПЛОМНА РОБОТА

РОЗРОБКА АНТИВІРУСНОЇ ПРОГРАМИ ДЛЯ ОПЕРАЦІЙНОЇ СИСТЕМИ WINDOWS

Розробив: ст. групи КНТ-227
Керівник: доцент, к.т.н.

Мирончук А.А.
Сердюк С.М.

Рисунок Б.1 – Слайд №1

Мета роботи

розробка програмного забезпечення, яке працює по запиту користувача для виявлення шкідливих програм, шляхом сигнатурного сканування

Об'єкт дослідження – процеси захисту інформації в операційній системі (ОС) Windows

Предмет дослідження – методи захисту інформації від вірусів в ОС Windows



Задачі, що вирішуються

- огляд типів та характеристик вірусу;
- огляд існуючих антивірусних програм;
- розробка технічного завдання;
- розробка архітектури системи;
- розробка алгоритму роботи сканеру;
- програмна реалізація проєктів застосунку;
- розробка необхідної документації.



НУ “ЗП”, Кафедра програмних засобів

Рисунок Б.3 – Слайд №3

3

Приклад опису одного з вірусів

VirusShare.com - Because Sharing is Caring

[Home](#) • [Hashes](#) • [Research](#) • [About](#)

Please [login](#) to search and download.

System currently contains 36,718,200 malware samples.

Report for a sample recently added to the system:
508cd3d530af3a57ff5eb5ba61b07d436e792da6d72c4dabe7818c251bb52cdd

VirusShare info last updated 2020-12-22 00:01:03 UTC

MD5	e672a52267090c808e2539815fce2c23			
SHA1	0b33accb74dc0964520d9309233025d30b4a66e1			
SHA256	508cd3d530af3a57ff5eb5ba61b07d436e792da6d72c4dabe7818c251bb52cdd			
SSDeep	3072.6Q4SPZD3UcjvG8rM2hFcXmNRSFI4t+rBKMLZosBbt2HrCFY0ZRxiYHaxkB.UwXmNRKBfp			
Size	197,845 bytes			
File Type	HTML document, UTF-8 Unicode text, with very long lines			
Mime Type	text/html			
Extension	html			
TrID	Atom web feed (92.2%) HyperText Markup Language (7.7%)			
Detections (2/62)	Fortinet JS/ScriptP.I.Nftr Microsoft Trojan.ScriptWacatac.Blm1 VirusTotal Report submitted 2020-11-30 14:19:19 UTC			
Exif Data	ContentType text/html, charset=UTF-8 FileSize 193 kB FileType HTML FileTypeExtension html Generator blogger MimeType text/html Title Feliz Natal! ~ Taverna do Elfo e do Arcanos viewport width=device-width, initial-scale=1.0			

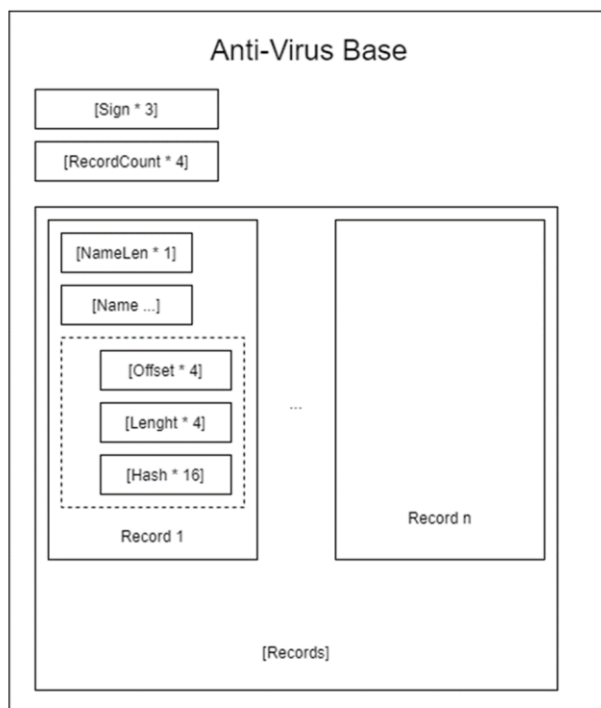


НУ “ЗП”, Кафедра програмних засобів

Рисунок Б.4 – Слайд №4

4

Загальна структура файлу антивірусної бази



НУ "ЗП", Кафедра програмних засобів

5

Рисунок Б.5 – Слайд №5

Порівняльний аналіз мов для створення сканеру

Основні критерії оцінки:

1. Доступність функціоналу Win32 API.
2. Можливість створення DLL.
3. Приведення типів Win32.
4. Поєднання бібліотеки та оболонки.
5. Додаткова робота з файлами.
6. Робота з адресою.
7. Робота з пам'яттю.
8. Робота з проміжними даними.
9. Можливості кодування даних.
10. Швидкість розробки

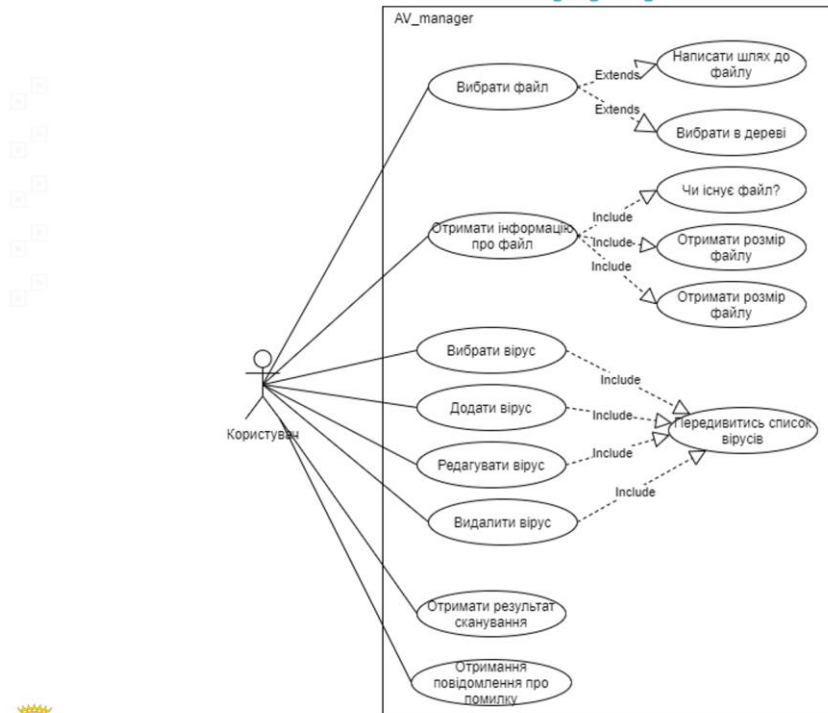


НУ "ЗП", Кафедра програмних засобів

6

Рисунок Б.6 – Слайд №6

Діаграма прецедентів менеджера антивірусу

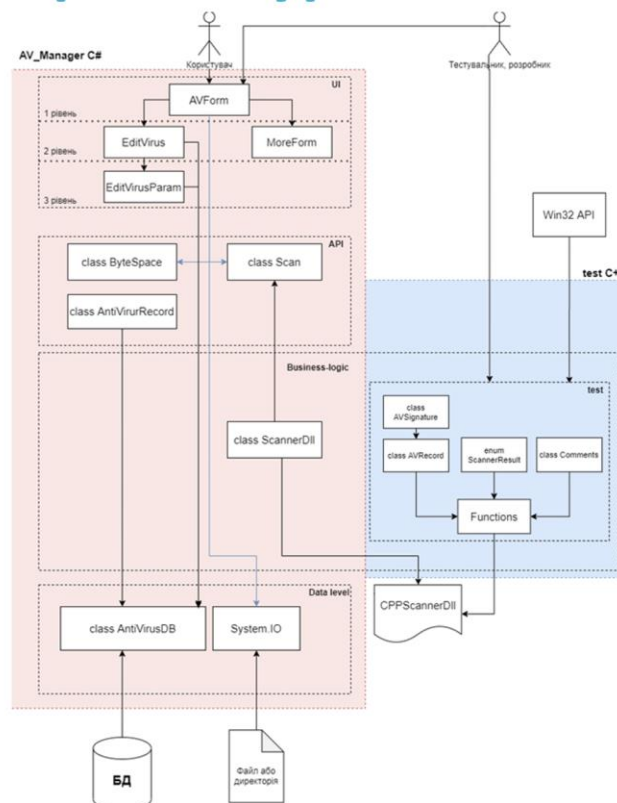


НУ "ЗП", Кафедра програмних засобів

7

Рисунок Б.7 – Слайд №7

Архітектура системи

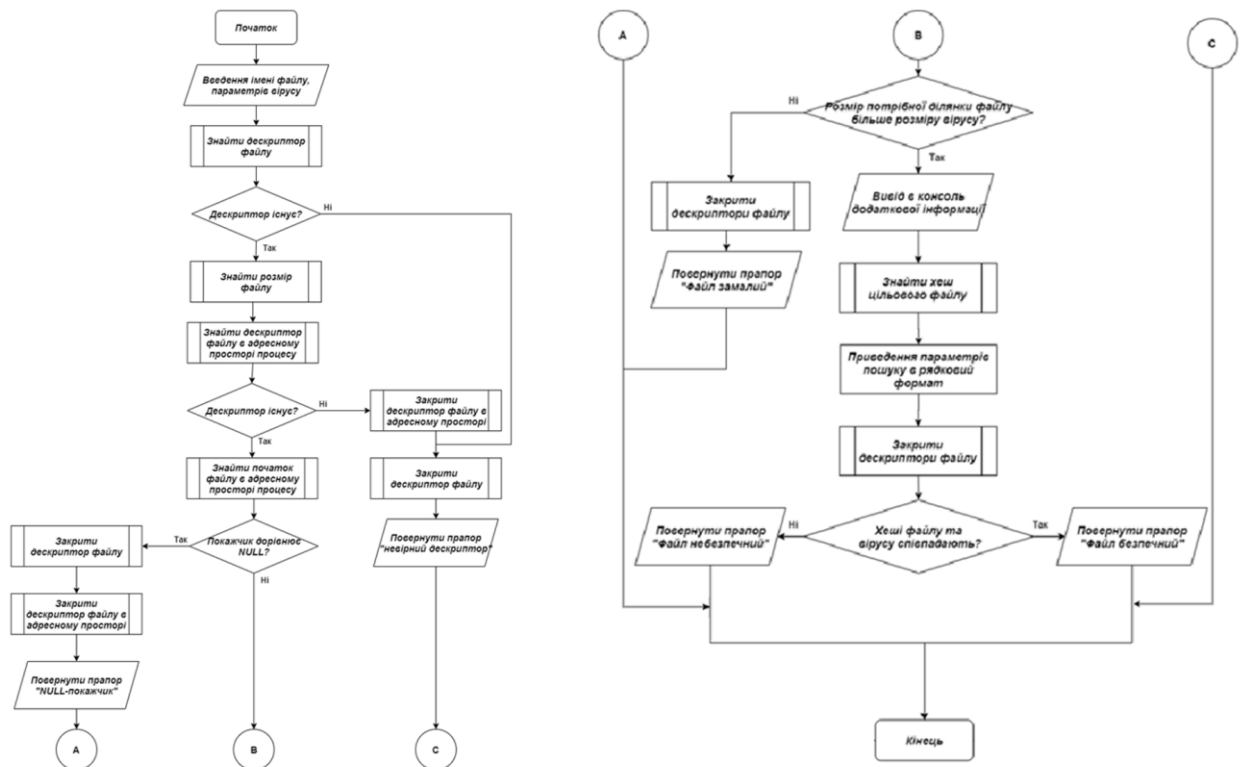


НУ "ЗП", Кафедра програмних засобів

8

Рисунок Б.8 – Слайд №8

Блок-схема алгоритму сканера

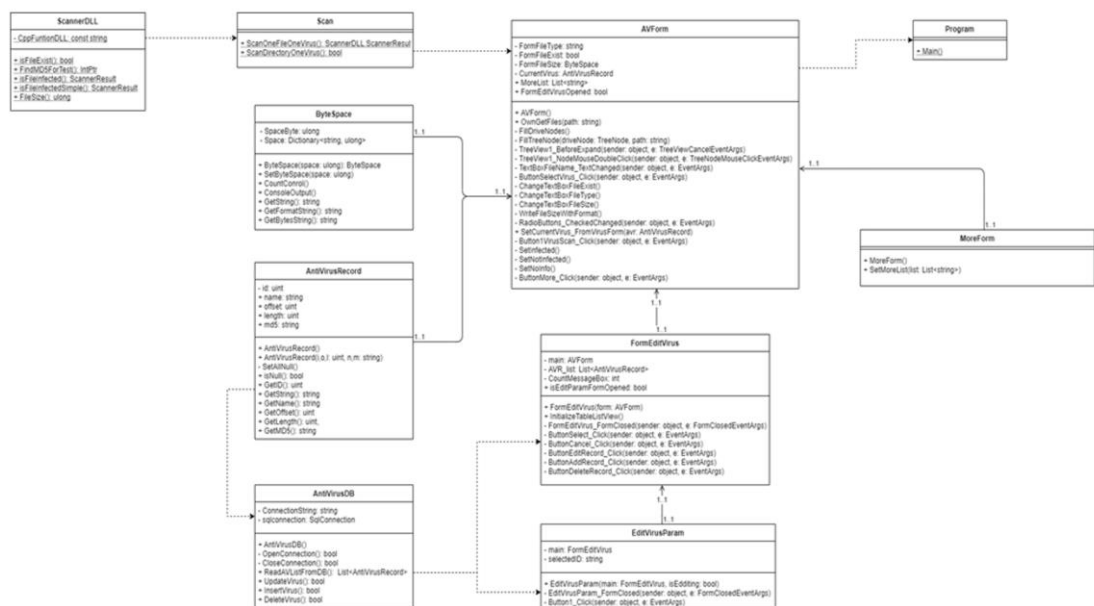


НУ "ЗП", Кафедра програмних засобів

Рисунок Б.9 – Слайд №9

9

Діаграма класів проекту AV_Manager



НУ "ЗП", Кафедра програмних засобів

Рисунок Б.10 – Слайд №10

10

Методи класу Scan

Методи класу	Опис
<i>public:</i>	
public static ScannerDLL.ScannerResult ScanOneFileOneVirus(string filename, AntiVirusRecord avr, out string h1, out string h2, out string FSM, out string FH)	Функція реалізує класичну перевірку конкретного файлу на наявність одного конкретного вірусу. За результатом такої перевірки, функція також повертає і додаткові дані про дескриптори файлу, про адрес початку файлу, а також про його контрольну суму.
public static bool ScanDirectoryOneVirus(string filename, AntiVirusRecord avr, out List<string> ResultList)	Функція реалізує перевірку конкретної директорії на наявність одного конкретного вірусу. Додатково, функція повертає лог перевірки: список з ім'ям вірусу, ім'ям файлу директорії, результатом цієї попарної перевірки.



НУ "ЗП", Кафедра програмних засобів

11

Рисунок Б.11 – Слайд №11

Використані функції Win32 API

Застосування методу у програмі	Опис
HANDLE hfile = CreateFileA(filename, FILE_READ_ACCESS, NULL, NULL, OPEN_EXISTING, NULL, NULL)	Функція створює або відкриває каталог, фізичний диск. Функція повертає дескриптор, який може бути використаний для доступу до об'єкта.
DWORD file_size = GetFileSize(hfile, NULL)	Функція видобуває розмір зазначеного файлу. Розмір файлу, який може бути повідомлений за допомогою цієї функції обмежується значенням подвійного слова (DWORD).
HANDLE hMap = CreateFileMappingA(hfile, NULL, PAGE_READONLY, NULL, file_size, NULL)	Функція створює або відкриває іменований або безіменний об'єкт для заданого файлу.
LPVOID File = MapViewOfFile(hMap, FILE_MAP_READ, NULL, NULL, file_size)	Функція відображає представлення файлу в адресний простір визначеного процесу.
return GetFileAttributesA(filename) != INVALID_FILE_ATTRIBUTES	Функція повертає встановлені атрибути в стилі файлової системи FAT визначеного файлу або каталогу. В даному випадку перевіряємо, чи існує файл.
CloseHandle(hfile)	Функція закриває дескриптор відкритого об'єкта.



НУ "ЗП", Кафедра програмних засобів

12

Рисунок Б.12 – Слайд №12

Сумісність з версіями Windows

Модулі OC	WIN32Ari (системні функції)	Бібліотеки C++ (нижній рівень програми – антивірус)	Бібліотеки C# (верхній рівень програми – інтерфейс користувача)
Windows 10	Так	Так	Так
Windows 8	Так	Так	Так
Windows 7	Так	Так	Так
Windows XP	Так	Так	Hi
Windows 2000 Professional	Так	Hi	Hi
Windows 98	Так	Hi	Hi
Windows 95	Так	Hi	Hi



НУ “ЗП”, Кафедра програмних засобів

13

Рисунок Б.13 – Слайд №13

Головна форма програми

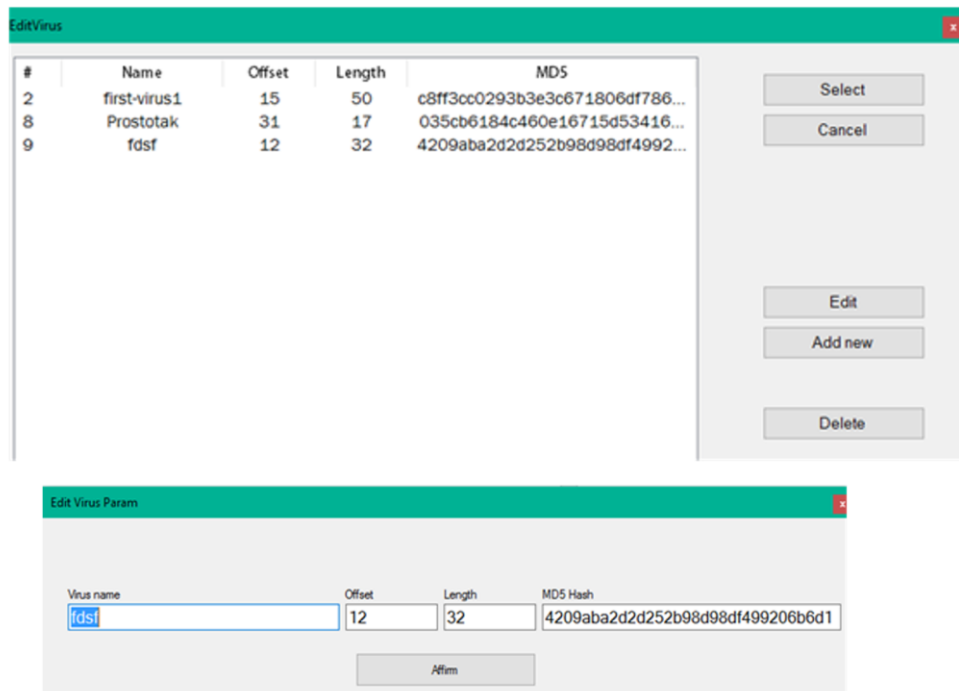


НУ “ЗП”, Кафедра програмних засобів

14

Рисунок Б.14 – Слайд №14

Форма редагування антивірусної бази

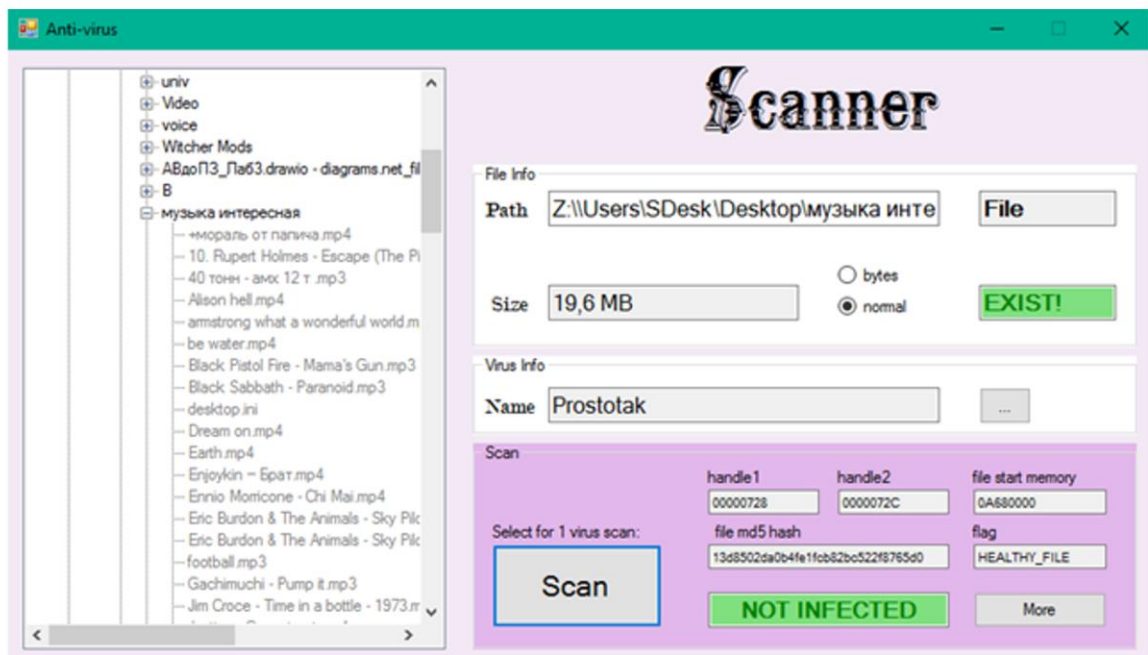


НУ "ЗП", Кафедра програмних засобів

15

Рисунок Б.15 – Слайд №15

Приклад результатів сканування



НУ "ЗП", Кафедра програмних засобів

16

Рисунок Б.16 – Слайд №16

Висновки

- проведено огляд типів та характеристик вірусу
- проведено огляд існуючих антивірусних програм;
- розроблено технічне завдання;
- розроблено архітектуру системи;
- розроблено алгоритм роботи сканеру;
- розроблено програмні проекти застосунку;
- розроблено необхідну документацію.



НУ "ЗП", Кафедра програмних засобів

Рисунок Б.17 – Слайд №17

17

**Доклад закінчено,
дякую за увагу!**



НУ "ЗП", Кафедра програмних засобів

Рисунок Б.18 – Слайд №18

18