Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

# ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
**за першим (бакалаврським) рівнем вищої освіти**
на тему
**“Програмне забезпечення системи підвищення надійності функціонування ЦОД”**



Виконав здобувач вищої освіти
IV курсу, групи КІ-21-2
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Голяга Д.Р.
«____» _____ 2025 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Марченко К.М.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

**Центральноукраїнський національний технічний університет**

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“*Інформаційні технології*”*_____

Спеціальність *123 “*Комп’ютерна інженерія*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Комп’ютерна інженерія*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2025 року

# ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

*Голязі Денису Руслановичу*
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи підвищення надійності функціонування ЦОД*

2. Керівник роботи *Марченко Костянтин Миколайович, канд. техн. наук, доцент*
( прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 47-02 від 17.01.2025 року

3. Строк подання студентом роботи до захисту *23.05.2025 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи підвищення надійності функціонування ЦОД*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

*1. Призначення та область використання.*

*2.Перегляд аналогічних існуючих систем.*

*3. Опис і обґрунтування проектних рішень.*

*4. Етапи програмування системи.*

*5. Впровадження системи в промислову експлуатацію.*

*6. Висновки*

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

| | |
|---|---|
| *Структурна схема системи* | *1 аркуш* |
| *Функціональна схема системи* | *1 аркуш* |
| *Діаграма процесів* | *1 аркуш* |
| *Блок-схема алгоритму роботи додатку* | *2 аркуша* |

7. Дата видачі завдання « 17 » січня 2025 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти | Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти | Примітка |
|---|---|---|---|
| 1. | Аналіз існуючих систем | 10.03.2025 р. | |
| 2. | Постановка задачі, оформлення ТЗ | 15.03.2025 р. | |
| 3. | Розробка моделі компонента | 20.03.2025 р. | |
| 4. | Розробка структур даних | 25.03.2025 р. | |
| 5. | Розробка алгоритмів зв’язку та відображення | 30.03.2025 р. | |
| 6. | Програмування алгоритмів | 10.04.2025 р. | |
| 7. | Оформлення ПЗ | 17.04.2025 р. | |
| 8. | Попередній захист роботи | 23.05.2025 р. | |
| | | | |
| | | | |
| | | | |
| | | | |



Дата видачі завдання
« 17 » січня 2025 р.

Підпис керівника
_____ Марченко К.М.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Голяга Д.Р.
(прізвище та ініціали)

# АНОТАЦІЯ

**Голяга Д.Р. Програмне забезпечення системи підвищення надійності функціонування ЦОД. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.**

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи підвищення надійності функціонування ЦОД.

Метою розробки є програмне забезпечення системи підвищення надійності функціонування ЦОД.

Результат роботи – програмна реалізація системи підвищення надійності функціонування ЦОД.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ з ОС Windows 10/11.

Програму розроблено в середовищі Visual C#.

**Ключові слова**: комп'ютерна інженерія, підвищення надійності функціонування ЦОД

# ABSTRACT

**Golyaga D.R. Software for the system of increasing the reliability of the functioning of the data center. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.**

In this final qualification work for the first (bachelor's) level of higher education, software has been developed, which is intended for the system of increasing the reliability of the functioning of the data center.

The purpose of the development is software for the system of increasing the reliability of the functioning of the data center.

The result of the work is the software implementation of the system of increasing the reliability of the functioning of the data center.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with the software are provided.

The program can be used on a PC with Windows 10/11.

The program was developed in the Visual C# environment.

**Keywords:** computer engineering, increasing the reliability of the functioning of the data center

**ЗМІСТ**

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

| | | |
|---|---|---|
| АБГШ | – | аддітивний білий гаусів шум |
| ЕОМ | – | електронно-обчислювальна машина |
| ІМС | – | інтегральна мікросхема |
| ARQ | – | автоматичний запит повторної передачі |
| CD-DA | – | Compact Disc Digital Audio |
| CIRC | – | Cross Interleaved Reed Solomon Code |
| EAB | – | Embedded Array Block, блок зосередженої пам'яті |
| ECC | – | error-correcting code, код корекції помилок |
| FEC | – | метод прямої корекції помилок |
| LDPC | – | коди Галлагера |
| NAK | – | негативне підтвердження |

# ВСТУП

**Актуальність теми**. Основне завдання центрів обробки даних полягає в забезпеченні ефективного функціонування інформаційних систем, що підтримують бізнес-процеси організації. Бар'єром на шляху реалізації концепції корпоративного ЦОД можуть стати помилки при проектуванні інженерних систем. У першу чергу ЦОД – це рішення для надійної, захищеної, швидкої і якісної роботи додатків. Створення ЦОД ділиться на два етапи: побудова інфраструктури й створення апаратно-програмного комплексу. Тобто це завжди комплексне рішення – інфраструктура, що включає будинки, спеціальні приміщення, кабельні системи, а також комп'ютерні й програмно-апаратні платформи для надійної, захищеної, швидкої і якісної роботи додатків. При створенні інфраструктури ЦОД вирішуються такі завдання, як забезпечення безперебійного енергопостачання й необхідних кліматичних параметрів (температура, вологість), організація СКС, систем пожежної й охоронної сигналізації, пожежогасіння й димовидалення, фізичної безпеки й ін. Побудова ЦОД має незаперечні переваги: доступність інформації 24x7x365, зниження витрат на експлуатацію, оптимізація використання фізичних і людських ресурсів, зниження складності інфраструктури, підвищення керованості, підвищення надійності. Інженерне забезпечення ЦОД саме сприяє зменшенню числа відмов і потенційних ризиків від зовнішніх впливів. При дотриманні всіх параметрів побудови інфраструктури ЦОД число відмов і витрати на ремонт прагнуть до нуля. При створенні ЦОД ми намагаємося мінімізувати кількість виробників ПЗ й «заліза», перелік використовуваних моделей, конфігурації встаткування й ПЗ, а також створити політики, що запобігають впровадження надлишкової кількості серверів.

**Мета й завдання дослідження.** Метою роботи є програмне забезпечення системи підвищення надійності функціонування ЦОД.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем підвищення надійності функціонування ЦОД.

– Дослідження системи підвищення надійності функціонування ЦОД.

– Програмна реалізація системи підвищення надійності функціонування ЦОД.

**Практична цінність отриманих результатів** полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі підвищення надійності функціонування ЦОД.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи підвищення надійності функціонування ЦОД, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

# 1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

## 1.1 Призначення системи

Система призначена для підвищення надійності функціонування ЦОД. При проектуванні системи гарантованого електропостачання ЦОД часом приділяють недостатню увагу до забезпечення підведення електроживлення: якщо найближча магістраль не може забезпечити необхідну живильну потужність, то система просто не буде працездатна. Крім того, є проблеми й при управлінні параметрами навколишнього середовища, пов'язані з підтримкою відповідного мікроклімату й контролем активного встаткування ЦОД.

Існує два підходи до побудови архітектури ЦОД. У першому все активне встаткування встановлюється у відкритих стійках, а контроль мікроклімату усередині приміщення здійснюється системами кондиціювання. Недолік цього підходу в тому, що він не може забезпечити потрібний мікроклімат у випадку установки активного встаткування більше 3,5 кВт на стійку. При другому підході все активне встаткування встановлюється в закриті шафи, при цьому потрібне додаткове встаткування, що здорожує створення центрів обробки даних. Втім, у цьому випадку можна ущільнювати встановлюване встаткування в шафах, знижуючи тим самим витрати на його обслуговування.

У сфері протипожежного захисту можна виділити два основні завдання. Перше завдання полягає в попередженні й оповіщенні й задимленні або пожежі усередині ЦОД завдяки звуковій, світловій сигналізації, мобільній сигналізації на телефон або пейджер, а також засобам віддаленого знеструмлення об'єкта. Друге завдання полягає в засобах пожежогасіння усередині ЦОД. При цьому враховуються особливості пожежестійких покриттів стелі, стін, підлоги, установка захищених дверей, засобів витяжки газу, апарати для подиху в аварійній ситуації й т.п.

Наявність високопродуктивного ЦОД є одним з основних конкурентних переваг сервісної геофізичної компанії. Основна проблема, з якою ми зіштовхуємося при проектуванні ЦОД, – це висока щільність розміщення серверів і необхідність управління більшими обсягами даних. Тому ми приділяємо саму серйозну увагу зниженню енергоспоживання системи й зменшенню надлишкових витрат енергії, а також пошуку ефективних рішень із погляду співвідношення: енергоспоживання – продуктивність і продуктивність – ємність. Це й системи кондиціювання, і використання більше ефективних систем забезпечення електропостачанням, а також пошук ефективних рішень із погляду перспектив розвитку підприємства й розширення потужностей ЦОД.

### 1.2 Область застосування

Устрій IT-інфраструктури будь-якого підприємства повинне вестися з урахуванням постійного дотримання високого рівня безпеки й доступності серверів. Пошук оптимального рішення цього завдання приводить великий і середній бізнес до того, що обчислювальні ресурси поступово акумулюються в спеціалізованих центрах обробки даних (ЦОД), що представляють собою гнучке середовище, здатну при необхідності перерозподіляти навантаження між виконуваними завданнями й успішно справлятися з піковими потоками даних. Безперебійне функціонування подібних центрів нерозривно пов'язане з надійністю їхньої інженерної інфраструктури, а зокрема – системи електропостачання.

Щоб уникнути розпливчастих формулювань як при формуванні технічного завдання до споруджувана дата-центрам, так і при описі характеристик уже функціонуючих ЦОД, була уведена незалежна система класифікації Tier. Відповідно до цієї градації (що входить у стандарт TIA/ EIA-942), всі ЦОДи розділяються по чотирьох класах, виходячи з відказостійкості встаткування.

Рівні Tier 1 і Tier 2 мають на увазі порівняно низьке резервування[1] систем ЦОД (допускається навіть його відсутність). Головна особливість дата-центрів таких класів полягає в необхідності зупинки надання сервісів для проведення будь-яких профілактичних або ремонтних робіт. Перевага ЦОД рівнів Tier 1 і Tier 2 – порівняно невисока вартість обчислювальних ресурсів.

Дата-центри класу Tier 3 і Tier 4 принципово відрізняються від об'єктів більше низького рівня тим, що в них є достатнє резервування як IT, так і інженерних систем, щоб здійснювати ремонтні роботи, не зупиняючи запущені сервіси. Між собою рівні 3 і 4 (як, втім, і рівні 1 і 2) розрізняються припустимим коефіцієнтом відказостійкості. Для ЦОД рівня Tier 3 він становить 99,749%, а для Tier 4 – 99,982%. Очевидно, що останній забезпечує вкрай надійну структуру дата-цента – досвідченим шляхом відмови встаткування зменшені до одного чотиригодинного випадку в п'ятирічний період.

Природно, якщо говорити про великий і середній бізнес, найбільш затребуваними виявляються дата-центри класу Tier 3 і Tier 4. Для досягнення настільки високого рівня надійності сучасні технології пропонують цілий ряд інструментів, що працюють на всіх трьох логічних рівнях ЦОД: програмному, телекомунікаційному й інженерному.

Центри даних класу Tier IV найбільше повно відповідають концепціям відказостійкості комп'ютерного встаткування, у якому використовується кластеризація ЦПУ, масиви RAID DASD і резервовані канали передачі даних, що забезпечують високу надійність, експлуатаційну готовність і ремонтопридатність.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи підвищення надійності функціонування ЦОД, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

# 2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

## 2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

У даному розділі розглянемо програми які дозволяють здійснювати завадостійке кодування, при запису файлу на диск, для підвищення надійності його зберігання.

**ICE ECC**

ICE ECC – це утиліта для перевірки й відновлення ушкоджених файлів. ICE ECC дозволяє захистити важливі файли від ушкодження використовуючи коди циклічного коду.

Типові CD-R/DVD-R диски починають деградувати вже після декількох років зберігання. Перш, ніж записати дані на CD-R/DVD-R, необхідно захистити їх від ушкодження за допомогою ICE ECC.

ICE ECC легко використовувати. Щоб захистити файли від ушкодження, необхідно виділити файли й натиснути "Create". ICE ECC створить файл/файли з кодами корекції помилок (.ecc) для виділених файлів. Можна легко контролювати розмір і кількість файлів *.ecc.

Коли потрібно перевірити, чи були файли модифіковані або ушкоджені, просто виберіть .ecc файл і натисніть "Verify". ICE ECC зробить аналіз цілісності й повідомить, якщо файли були ушкоджені або втрачені. У цьому випадку ICE ECC зможе автоматично відновити їх.

ICE ECC пропонує нові технології для захисту файлів:

1. ICE ECC може захищати не тільки окремі файли. ICE ECC може працювати з каталогами теж. Можна легко захистити весь CD-R або DVD-R диск або будь-які файли або каталоги від ушкоджень.

2. Коди циклічного коду вимагають потужний процесор для обчислень. ICE ECC забезпечує найшвидшу реалізацію кодів циклічного коду у світі.

3. ICE ECC має повну підтримку Unicode. Це означає, що файли з будь-якими іменами файлів можуть бути захищені від ушкодження.

4. ICE ECC не має обмежень на кількість або розмір файлів, що захищаються.

5. ICE ECC використовує розподілений механізм для зберігання файлів з кодами корекції помилок (.ecc). Відновлення інформації можливо навіть, якщо файли з кодами корекції помилок сильно ушкоджені.

6. ICE ECC використовує спеціальну компресію для зберігання каталогу файлів. Це забезпечує мінімальний размір.ecc файлів навіть для більших колекцій файлів, що захищаються.

7. ICE ECC може не тільки відновлювати ушкоджені файли. ICE ECC може відновити втрачені файли, якщо їхній розмір менше, ніж розмір всіх файлів файлів з кодами корекції помилок.

8. ICE ECC використовує алгоритм, що дозволяє знаходити зрушені дані незалежно від відстані зрушення й буде працювати із блоками будь-якого розміру. Як .ecc файли, так і файли даних не чутливі до зрушення.

9. ICE ECC підтримує черги для будь-яких операцій.

10. ICE ECC підтримує мультиядерні процесори або багатопроцессорні системи.

11. Є повна підтримка командного рядка в ICE ECC. Команди з командного рядка автоматично додаються в чергу й виконуються асинхронно.

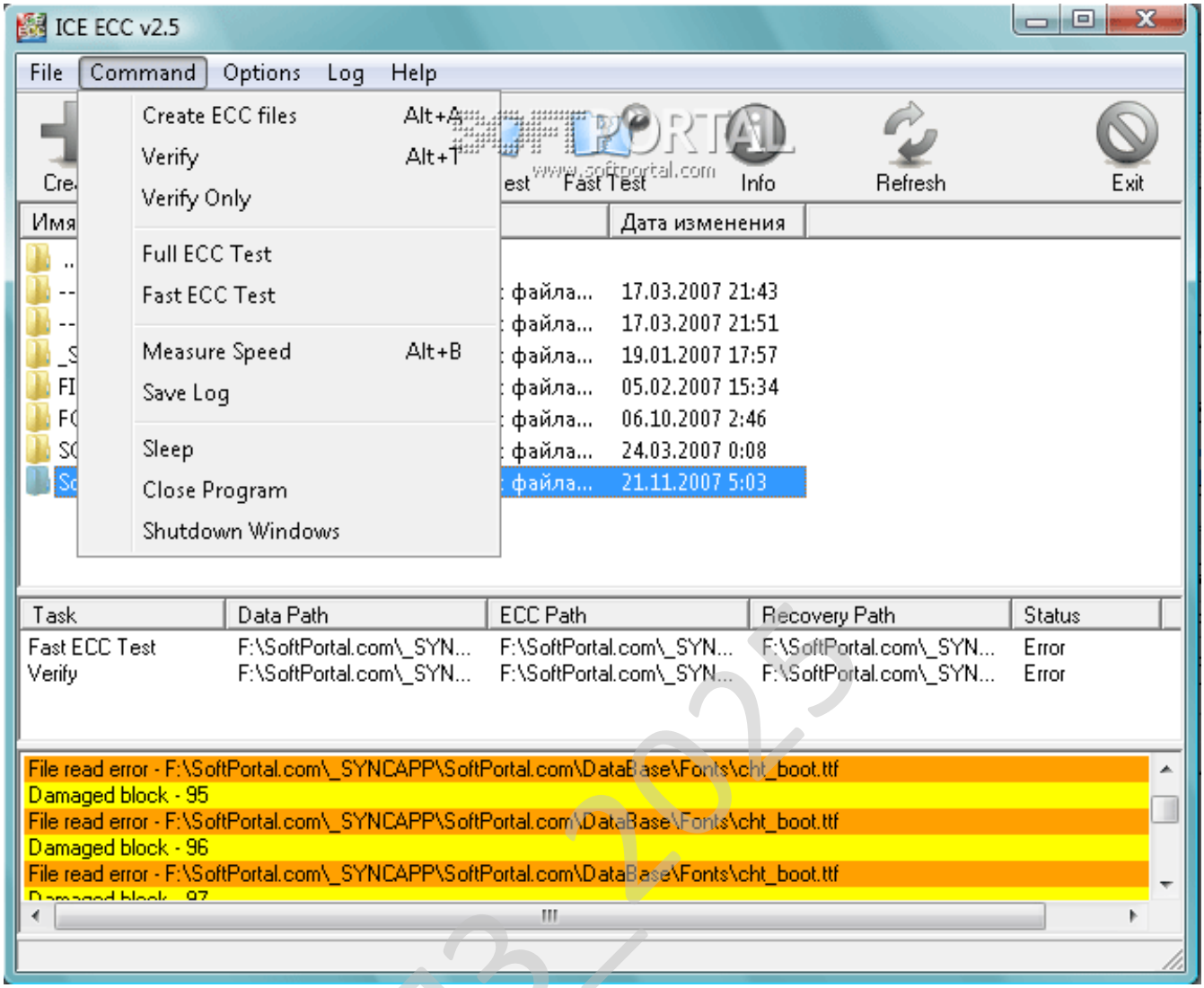


Рисунок 2.1 – Інтерфейс користувача ICE ECC

Є інші програми, схожі на ICE ECC: PAR2, QuickPar, і інші. Але ICE ECC перевершує їх всіх не тільки тим, що вміє працювати з каталогами. Результати тестів скажуть самі за себе:

Test A: CPU – AMD Athlon X2, 45 файлів – 627 225 428 байт – 1024 блоків – 256 блоків надлишкової інформації.

Test B: CPU – AMD Athlon X2, 6571 файлів – 616 439 775 байт – 1024 блоків – 256 блоків надлишкової інформації.

Таблиця 2.1 – Результати тесту A

| | Create | Create + Fast Test |
|---|---|---|
| PAR2 | 07:27 | |
| QuickPar | | 03:14 |
| ICE ECC – 1 CPU mode | 02:51 | 02:58 |
| ICE ECC – 2 CPU mode | 02:30 | 02:37 |

Таблиця 2.2 – Результати тесту B

| | Create | Create + Fast Test |
|---|---|---|
| PAR2 | Failed: Block count is too small | |
| QuickPar | Failed: Too many files selected | |
| ICE ECC – 2 CPU mode | 02:27 | 02:38 |

**QuickPar**

QuickPar – це програма, що використовує алгоритм виправлення помилок циклічного коду для перевірки цілісності файлу або групи файлів. Вона часто використовується для відновлення файлів, завантажених через Usenet.

Ключові особливості й характеристики QuickPar:

– При відзначеній функції "Автовідновлення", неправильно названі або ушкоджені файли (з усіма доступними блоками) будуть негайно відновлені.

– Діалогове вікно "параметри" тепер має прапорці "Відновити автоматично" і "Перевіряти автоматично". Ці функції контролюють, як у вікні перевірки встановлені параметри "Автовідновлення" і "Моніторинг", коли ви в перший раз відкриваєте файл PAR2.

– Тепер при створенні файлів PAR2, програма QuickPar може створювати розділені версії вихідних файлів.

– При створенні файлів PAR2 або при їхньому використанні для відновлення, програма QuickPar буде запитувати підтвердження, якщо ви натиснете на кнопку "Стоп".

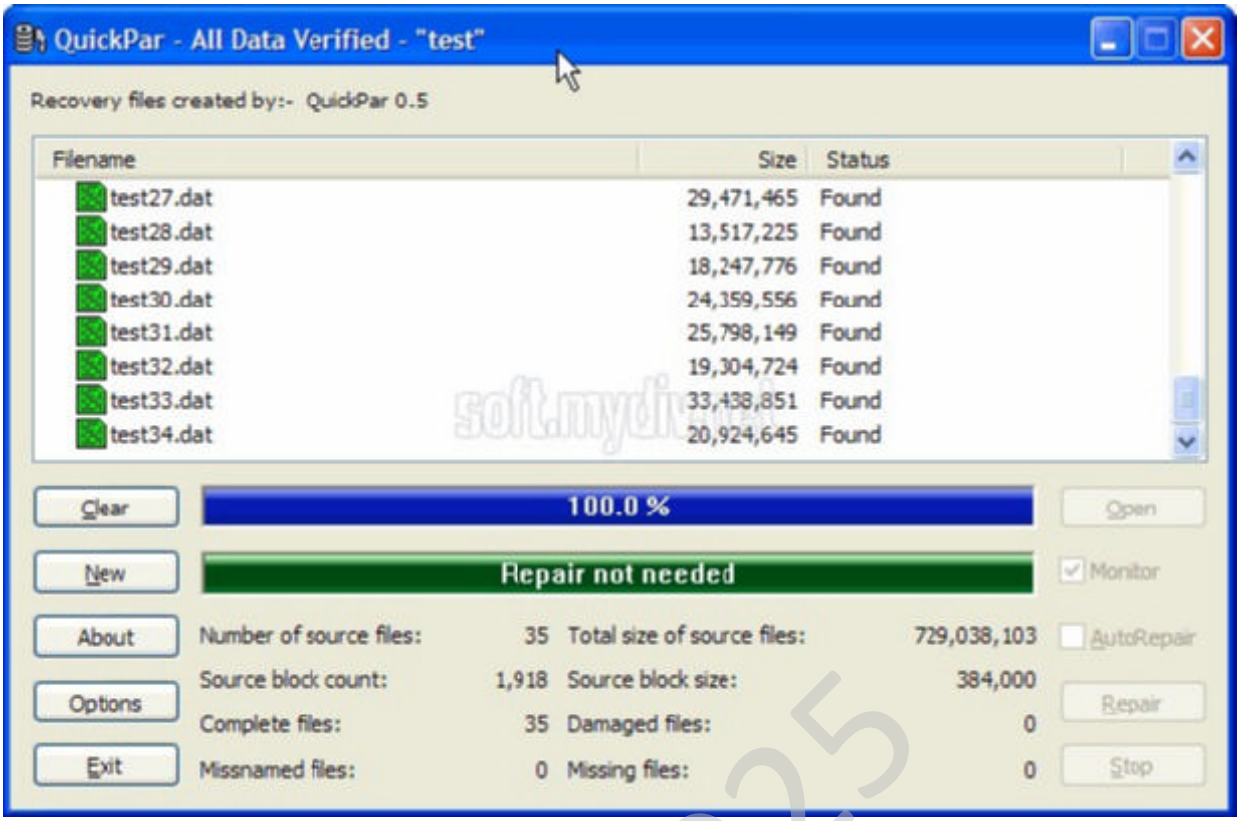


Рисунок 2.2 – Інтерфейс користувача QuickPar

– При перевірці файлів програма QuickPar буде автоматично розпізнавати неповні версії файлів, створені за допомогою yDec, з розширенням "(nnn).tmp".

– При перевірці файлів, можна буде, клікнувши правою кнопкою миші по файлі в списку, одержати доступ до контекстного меню оболонки, що відповідає тому ж файлу. Також ви зможете подвійним клацанням по файлі виконати дію за замовчуванням (звичайно, таким чином, файл відкривається).

– Після успішної перевірки файлу (або його відновлення), тепер можна буде створювати додаткові файли PAR2, нажавши на кнопку "Додатковий файл". Програма QuickPar відобразить діалогове вікно для створення файлів PAR2, але всі настроювання, крім обраних і, будуть деактивовані.

– Після успішного відновлення, програма QuickPar буде видаляти ушкоджені або неповні файли даних, а також файли PAR2, які більше не потрібні. Програма QuickPar завжди буде зберігати невеликий "пріоритетний" файл PAR2. Цю функцію можна настроїти в діалоговому вікні "Параметри".

## 2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Програмне забезпечення написане мовою Visual C#. Ця мова обрана виходячи з наступних міркувань. Visual C# – строго типізована об'єктно-орієнтована мова, призначена для розробки різноманітних безпечних і потужних додатків, виконуваних у середовищі .NET Framework. Мовою Visual C# можна розробляти звичайні клієнтські додатки Windows, веб-служби XML, розподілені компоненти, додатки типу " сервер-клієнт", додатки баз даних і багато яких інших. В Visual C# 2012 є розширений редактор коду, конструктори зі зручним користувальницьким інтерфейсом, вбудований відладник і багато інших засобів, покликані спростити розробку додатків мовою Visual C# версії 5.0 і .NET Framework версії 4.5.

Синтаксис Visual C# дуже виразний, але простий у вивченні. Усі, хто знаком з мовами C, C++ або Java з легкістю взнають синтаксис із фігурними дужками, характерний для мови Visual C#. Розроблювачі, що знають кожну із цих мов, як правило, зможуть домогтися ефективної роботи з мовою Visual C# за дуже короткий час. Синтаксис Visual C# робить простіше те, що було складно в C++, і забезпечує потужні можливості, такі як типи значень Nullable, перерахування, делегати, лямбда-вираження й прямий доступ до пам'яті, чого немає в Java. Visual C# підтримує універсальні методи й типи, забезпечуючи більше високий рівень безпеки й продуктивності, а також ітератори, що дозволяють при реалізації колекцій класів визначати власне поводження ітерації, що може легко використовуватися в клієнтському коді. В Visual C# 5.0 вираження LINQ (Language-Integrated Query) роблять строго-типізований запит першокласною конструкцією мови.

Як об'єктно-орієнтована мова, Visual C# підтримує поняття інкапсуляції, спадкування й поліморфізму. Всі змінні й методи, включаючи метод `Main` – крапку входу додатка – інкапсулюється у визначення класів. Клас може

успадковувати безпосередньо з одного родового класу, але може реалізовувати будь-яке число інтерфейсів. Для методів, які перевизначають віртуальні методи в батьківському класі, необхідно ключове слово override, щоб виключити випадкове повторне визначення. У мові Visual C# структура схожа на полегшений клас: це тип, що розподіляється по стопках, що реалізує інтерфейси, але не підтримуюче спадкування.

На додаток до основних описаних об’єктно-орієнтованих принципів, мова Visual C# спрощує розробку компонентів програмного забезпечення завдяки декільком інноваційним конструкціям мови, у число яких входять наступні:

– Інкапсульовані підписи методів, називані делегатами, які підтримують строго-типізовані повідомлення про події.

– Властивості, що виступають у ролі методів доступу для закритих змінних-членів.

– Атрибути з декларативними метаданими про типи під час виконання.

– Вбудовані коментарі XML-документації.

– LINQ (Language-Integrated Query), що пропонує вбудовані можливості запитів у різних джерелах даних.

Якщо буде потрібно забезпечити взаємодію з іншим програмним забезпеченням Windows, таким як об'єкти COM або власні бібліотеки DLL Win32, у мові Visual C# можна використовувати процес, що називається "Interop". Процес Interop дозволяє програмам на Visual C# виконувати практично будь-які дії, які може виконувати вихідний додаток на C++. Мова Visual C# підтримує навіть покажчики й поняття "небезпечного" коду для тих випадків, коли прямий доступ до пам'яті має вкрай важливе значення.

Процес побудови Visual C# у порівнянні з C і C++ простий і є більше гнучким, чим в Java. Немає окремих файлів заголовка, а методи й типи не потрібно повідомляти в певному порядку. У вихідному файлі Visual C# може бути визначене будь-яке число класів, структур, інтерфейсів і подій.

**Архітектура платформи .NET Framework**

Програма мовою Visual C# виконується в середовищі .NET Framework – інтегрованому компоненті Windows, що містить віртуальну систему виконання (середовище CLR) і уніфікований набір бібліотек класів. Середовище CLR являє собою комерційну реалізацію корпорацією Майкрософт інфраструктури CLI, що є міжнародним стандартом, який лежить в основі створення середовищ виконання й розробки, у яких забезпечується тісна взаємодія між мовами й бібліотеками.

Вихідний код, написаний мовою Visual C#, компілюється в проміжну мову (IL) у відповідності зі специфікацією CLI. Код IL і ресурси, такі як растрові зображення й рядки, зберігаються на диску у файлі, що виконується, називаному складанням, з розширенням EXE або DLL у більшості випадків. Складання містить маніфест із відомостями про типи складання, версії, мови й регіональні параметри та вимоги безпеки.

При виконанні програми на Visual C# складання завантажується в середовище CLR залежно від відомостей у маніфесті. Далі, якщо вимоги безпеки дотримані, середовище CLR виконує JIT-компіляцію для перетворення коду IL в інструкції машинного коду. Середовище CLR також надає інші служби, що відносяться до автоматичного збору сміття, обробки виключень і управлінню ресурсами. Код, виконуваний середовищем CLR, іноді називають "керованим кодом" у протиставлення "некерованому коду", що компілюється в машинний код, призначений для певної системи. Далі показані відносини під час компіляції й час виконання між файлами з вихідним кодом Visual C#, бібліотеками класів .NET Framework, складаннями й середовищем CLR.

Взаємодія між мовами є ключовою особливістю .NET Framework. Оскільки код IL, створюваний компілятором Visual C# відповідає специфікації CTS, код IL на основі Visual C# може взаємодіяти з кодом, створюваним версіями мов Visual Basic, Visual C++, Visual J# платформи .NET Framework і ще більш ніж 20 CTS-сумісних мов. В одному складанні може бути кілька модулів, написаних

на різних мовах платформи .NET Framework, і типи можуть посилатися один на одного, як якби вони були написані на одній мові.

Крім служб часу виконання, в.NET Framework також є велика бібліотека, що складається з більш ніж 4000 класів, організованих по просторах імен, які забезпечують різноманітні корисні функції для будь-яких дій, починаючи від введення й виведення файлів для управлінням рядками для розбивки XML, і закінчуючи елементами управління Windows Forms. У звичайному додатку мовою Visual C# бібліотека класів .NET Framework інтенсивно використовується для "устрою" коду.

### 2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи підвищення надійності функціонування ЦОД.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

# 3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

## 3.1 Опис функціонування системи

Центр обробки даних – це складний комплекс, що включає в себе обчислювальні потужності, елементи IT – інфраструктури, будівельних і інженерних систем, основними функціями якого є – зберігання, обробка й передача інформації. У ЦОД на відносно невеликій площі зосереджені потужні обчислювальні ресурси:

– сервера й системи зберігання даних (СЗД), що здійснюють зберігання й обробку інформації;

– мережне встаткування, відповідальне за обмін даними усередині ЦОД, а також за зв'язок із зовнішніми споживачами;

– інженерні системи, системи безпеки, системи диспетчеризації й моніторингу, що забезпечують ефективну роботу й захист зосередженого в ЦОД обчислювального центра.

ЦОД можуть бути комерційними, призначені в основному для надання послуг co-location (оренди непохитно-місць, серверного встаткування), а також корпоративними, створеними для підтримки внутрішнього інформаційного середовища компанії-замовника.

Очевидно, що зараз робота сучасного, конкурентно здатного підприємства неможлива без використання інформаційних систем, які підтримують життєво важливі процеси. Найчастіше неполадки в роботі, простий таких систем, їхня недостатня продуктивність може привести до фінансових збитків, втраті корпоративної інформації, нанести серйозну втрата іміджу компанії, лояльності клієнтів і партнерів. Крім того, питання безперервного функціонування інформаційного середовища особливо гостро встають перед зростаючими, що динамічно розвиваються компаніями. Адже в умовах росту неминуче виникають

проблеми збільшення обсягу й цінності інформації, росту збитку при порушенні працездатності інформаційних систем.

Для підприємств, де питання забезпечення надійності функціонування інформаційних систем є критично важливими необхідно створення правильно спроектованого й побудованого силами професійного системного інтегратора корпоративного центра обробки даних (ЦОД). Це гарантія забезпечення безперервної роботи всіх інформаційних систем компанії, а також надійний фундамент впровадження нових IT-розробок.

Найбільш затребуваними IT-технології завжди були на підприємствах, де робота інформаційних систем є найбільш критичною для бізнесу. До таких компаній у першу чергу ставляться державні структури, банки й телекомунікаційні компанії. Ці підприємства можна назвати зрілими, з погляду рівня розвитку IT, користувачами зі сформованою культурою, підходами й розумінням місця інформаційних технологій у бізнес-процесах компанії або організації. Для таких компаній характерні вже створені великі ЦОДи, оснащені найсучаснішим устаткуванням і програмним забезпеченням. У таких ЦОДах завжди є потреби розширення, підвищення надійності, мінімізації енерговитрат на функціонування ЦОД, витрат на обслуговування, сервіс.

Сьогодні ми можемо спостерігати, як керівництво компаній з інших галузей проявляє все більший інтерес до автоматизації, впровадженню IT – технологій у бізнесі. Усе більше «IT-жадібними» стають промислові підприємства, роздрібні торговельні мережі, страхові компанії. Саме тут спостерігається найбільший інтерес до IT, зокрема до ЦОДів. Для таких компаній характерне рішення наступних завдань: де краще розмістити свій центр обробки даних, які програмні засоби найбільше повно зможуть вирішити завдання, яка оптимальна апаратна платформа для роботи необхідних додатків.

Таким чином, критичність інформаційних технологій для бізнесу усвідомлюють гравці все більшого числа сегментів ринку, IT глибше проникають в економіку підприємств. Змінюється відношення до IT з боку Топ-менеджменту:

багато керівників бачать в інформаційних технологіях не просто ще одна конкурентна перевага – а основу росту й розвитку бізнесу.

**Основні вимоги до ЦОД**

Тому що від роботи ЦОД залежить надійність всієї інформаційної системи, рівень послуг, надаваних компанією своїм клієнтам, продуктивність праці співробітників створюваного на підприємстві корпоративному ЦОД пред'являються наступні вимоги:

– Забезпечення безперервності функціонування всіх інформаційних систем.

– Висока продуктивність обробки й передачі даних.

– Доступність даних і додатків.

– Надійність зберігання даних, висока ємність зберігання.

– Відказостійкість.

– Катастрофостійкість.

– Високий ступінь масштабованості всіх систем ЦОД (з обліком 5-10 років активного розвитку компанії).

Необхідно відзначити, що усе більше Замовники звертають увагу на фактор мінімізації сукупної вартості володіння системою, що складається з витрат на енергоспоживання, обслуговування, сервіс.

**Склад інженерних систем ЦОД**

**IT-інфраструктура**

– Високонадійне серверне встаткування.

– Системи зберігання даних.

– Системи резервного копіювання й відновлення даних.

– Інфраструктурне й прикладне ПЗ.

– Система передачі даних ЦОД.

**Інженерні системи**

– Системи загального, гарантованого й безперебійного електроживлення.

– Система вентиляції й кондиціювання.

– Захисне заземлення.

– Загальбудівельні рішення приміщень.

**Системи безпеки:**

– Інформаційна безпека.

– Відеоспостереження.

– Система контролю управління доступом (СКУД).

– Системи сигналізації й оповіщення.

– Система протипожежної безпеки й пожежогасіння.

**Системи моніторингу, управління й диспетчеризації ЦОД**

– Оповіщення про аварійні події.

– Облік споживання ресурсів.

– Протоколювання подій.

**IT-інфраструктура ЦОД**

IT-інфраструктура – це обчислювальні потужності ЦОД, її можна назвати серцем ЦОД, саме вона відповідає за продуктивність, доступність даних і додатків, швидкість передачі даних, їхнє зберігання. До складу IT-інфраструктури входять:

– Серверне встаткування.

– Система зберігання даних.

– Система передачі даних ЦОД.

Побудова сучасної IT-інфраструктури зараз неможливо без використання блейд-серверів як ядро IT-інфраструктури, а також застосування технологій віртуалізації, значно підвищувальну ефективність серверів. З погляду економічної доцільності, масштабованості, продуктивності ці технології можна назвати справжнім проривом. Зараз цей напрямок активно розвивається, лідери світового ринку серверного встаткування й програмного забезпечення постійно пропонують нові розробки в цьому напрямку.

Застосування даних технологій при побудові IT-інфраструктури дозволяє:

– зменшити вартість володіння (експлуатації) IT-інфраструктури, включаючи витрати на електрику, витрати на сервіс, обслуговування;

– збільшити ефективність використання існуючих потужностей устаткування;

– забезпечити підвищену доступність додатків;

– прогнозувати й планувати розширення IT-інфраструктури.

**Інженерні системи ЦОД**

Для підтримки роботи високопродуктивного, дорогого встаткування, що становить IT-інфраструктуру, у ЦОД необхідно створити комплекс інженерних систем.

**Система електропостачання**

Насамперед, це, звичайно, система електропостачання ЦОД: гарантованого й безперебійного електроживлення. Робота цих систем є критично важливою для функціонування ЦОД – адже непередбачені перебої електропостачання в міських електромережах, перегони напруги можуть привести до виходу з ладу встаткування, зупинці роботи життєво важливих процесів у компанії. Тривале ж відключення електроживлення може привести до того, що не буде працювати ні зв'язок, ні телекомунікації, ні охоронні системи.

**Система кондиціювання**

Інша, не менш значима система підтримки функціонування роботи IT-інфраструктури в ЦОД – це резервна, відказостійка система кондиціювання, адже перегрів – одна з найпоширеніших причин зупинки роботи серверів.

У ЦОД для побудови систем клімат-контролю використовуються промислові прецизійні кондиціонери, найчастіше з функцією гарячого резервування. Також зараз актуальним стало використання в таких кондиціонерах функції «free-cooling», застосування якої дає економію енергоспоживання до 30%.

**Системи безпеки ЦОД**

Центр обробки даних – це стратегічне місце в будь-якій компанії, де перебуває дороге встаткування, консолідована корпоративна інформація – найважливіший актив сучасного підприємства. Очевидно, що в ЦОД необхідне забезпечення контролю, регламентованого доступу осіб. У даному рішенні були застосовані інтегровані системи: відеоспостереження, СКУД, протипожежної безпеки.

Інформаційна безпека: антивірусний захист, спам-фільтр, захист від вторгнень.

**Системи моніторингу, управління й диспетчеризації ЦОД**

Важливим компонентом ЦОД є система диспетчеризації й моніторингу, що дозволяє створити в ЦОД централізований моніторинг параметрів систем і їхніх станів: напруги електроживлення, температури в стійках, температури повітря й т.д. У даній системі важливу частину займає підсистема оповіщення й прогнозування відмов устаткування, критичних ситуацій у ЦОДі. Сучасні системи диспетчеризації дозволяють організувати інформування обслуговуючого персоналу за допомогою всіх доступних інтерфейсів зв'язку: e-mail, sms, автоматичний дозвон до заданої групи абонентів. Все це знижує ризики виникнення аварійних ситуацій, а також час на відновлення працездатності систем. Крім того, система диспетчеризації дозволяє налагодити облік енергоспоживання, витрат на вентиляцію й кондиціювання, що веде до економії й зниження сукупної вартості володіння системою.

Найважливіші вимоги до системи безперебійного живлення – це вихідна потужність, час автономної роботи, наявність убудованого й/або зовнішнього байпаса й можливість нарощування потужності без переробки ІБП.

1. Програмне забезпечення для дата-центрів дозволяє піднятися на інший рівень абстракції, працювати з віртуальними системами, не прив'язаними до конкретної географічної крапки розміщення серверів. Іншими словами, ЦОД може являти собою як один об'єкт, так і кілька центрів по всій країні, об'єднаних

однієї IT-інфраструктурою. Це дозволяє підвищити надійність сервісів, у тому числі зберігання даних, за рахунок дублювання інформації в різних географічних крапках.

2. Телекомунікаційні канали дозволяють багаторазово повторювати підключення серверів до Інтернету, що гарантує доступність розміщених у ЦОД даних навіть при обриві зв'язку на одній з магістральних ліній.

3. Необхідної безпеки ЦОД неможливо досягти без відказостійкості вхідних у нього інженерних систем, зокрема – систем кондиціювання, безпеки й т.п. Саме тому рівні надійності Tier 3 і 4 пред'являють досить тверді вимоги до системи електропостачання.

Культура роботи з інженерними системами, а також застарілі комплектуючі породжують ряд проблем з реалізацією високих вимог до надійності в реальних умовах. На українському ринку зустрічаються ситуації, коли в ЦОДах велику увагу приділяється одним характеристикам на шкоду іншим. Наприклад, у деталях пророблені системи технічної безпеки, а система електроживлення не продумана на належному рівні.

Проблеми системи електропостачання ЦОД і методи їхнього рішення

**Проблема 1. Недостатня підготовленість інженерів-проектувальників**

Опис: Відсутність спеціалізованого досвіду часто приводить до того, що компанії використовують у своїй роботі ті ж принципи, що й при проектуванні систем електропостачання житлових будинків або звичайних адміністративних або промислових об'єктів. Падіння кваліфікації й компетентності проектувальників – загальноукраїнський тренд: ринок ЦОД по іменах знає дійсно професійних профільних інженерів, і число їх, на жаль, поки ще вкрай невелико. Крім цього, існує проблема «політичного» тиску на сам процес проектування, як правило, на шкоду технічним характеристикам і економічним показникам проекту. У цьому випадку, навіть якщо проектувальник і має відповідну кваліфікацію, він обмежений у прийнятті рішень. Тому результат звичайно завжди дорожче, ніж він міг би бути, причому більше стають як капітальні, так і

наступні операційні витрати. На додаток до цього, проекти розробляються й узгоджуються вкрай довго, весь процес по строках часто «перевалює» за кордон одного календарного року. За рубежем за те ж самий час середньостатистичний ЦОД на трохи мегават уже вводиться в експлуатацію.

Рішення: Єдине можливе рішення – залучення до будівництва об'єкта профільних фахівців і виробників спеціалізованого якісного встаткування.

Будь-який фахівець повинен чітко розуміти, що основне завдання при проектуванні ЦОД – забезпечити максимальний рівень доступності серверів при мінімальному показнику ефективності споживання електроенергії (PUE)[2]. Для цього необхідно виконувати 4 правила:

– Максимально скоротити число аварій (якісне встаткування, повна диспетчеризація на всіх рівнях).

– Звести до мінімуму час відновлення після аварії.

– Застосовувати рішення й устаткування, що дозволяють проводити обслуговування без відключень (резервування не менш N+1, втичні розподільні системи).

– Використовувати енергоефективні рішення й технології (частотні перетворювачі, компенсатори реактивної потужності й ін.).

**Проблема 2. Простої системи у випадку аварії**

Опис: Високі вимоги по відказостійкості ЦОД рівнів Tier 3 і Tier 4 ведуть до того, що простий варіант із ремонтом або профілактикою під час відключення електропостачання неприйнятний для дата-центрів цих класів. У той же час ніяке встаткування не застраховане від поломок, крім того, будь-яка інженерна система періодично вимагає проведення огляду й технічного обслуговування.

Рішення: У дата-центрах високого рівня надійності необхідно використовувати системи, що дозволяють оперувати окремими елементами без відключення від живильної мережі. Сьогодні існують так звані рішення втичного монтажу для системи електропостачання. Торік на українському ринку з'явилася модульна розподільна система Smissline TP з убудованими струмоведучими

шинами. У ній реалізована ідея живлення типових модульних апаратів безпосередньо із шини без додаткового кабельного підключення, що забезпечує підвищену надійність з'єднань системи. Тому дане встаткування орієнтоване на споживачів першої, у тому числі – особливої категорії електропостачання».

**Категорії електропостачання**

Відповідно до Правил будови електроустановок (ПБЕ), відносно забезпечення надійності електропостачання споживачі електроенергії розділяються на наступні три категорії:

**I категорія** – електроприймачі, переривання електропостачання яких може викликати: небезпека для життя людей, ушкодження дорогого встаткування, масовий шлюб продукції, розлад технологічного процесу, порушення функціонування особливо важливих елементів комунального господарства.

**II категорія** – електроприймачі, переривання електропостачання яких приводить до масової недовідпустки продукції, масовим простоям робітників, механізмів і промислового транспорту, порушенню нормальної діяльності значної кількості міських і сільських жителів.

**III категорія** – всі інші електроприймачі, що не підходять під визначення I і II категорій.

**Проблема 3. Складності при розширенні й модернізації**

Опис: Модернізація встаткування – загальна проблема ЦОД, що не залежить від локальних особливостей географічного регіону, де він будується. Її джерелом є сам життєвий цикл дата-центра й величезна швидкість розвитку апаратного забезпечення. Середній час експлуатації ЦОД без капітальної модернізації IT-інфраструктури становить 10 років. Обчислювальні вузли, побудовані на серверах стандартної архітектури (X86), морально застарівають і вже не здатні задовольняти постійно зростаючий бізнес-вимогам після 3-5 років експлуатації. Обчислювальна техніка класу mid-range «живе» трохи довше – 5-7 років. Тільки встаткування класу high-end (мейнфрейми) може експлуатуватися

10 років і більше. Зміна парку встаткування неминуче спричиняє зміну існуючої системи розподілу електроживлення й в остаточному підсумку приводить до модернізації електропостачання ЦОДа.

Рішення: Єдине рішення, що дозволяє впоратися з постійним ростом потужності й необхідністю заміни встаткування або впровадження нових апаратів – побудова масштабованої системи електропостачання, що дозволяє підбудовувати її параметри під вимоги ІТ без витрат на переробку вже існуючих сегментів. Не завжди на початковому етапі побудови системи електропостачання очевидно, які компоненти знадобляться при розширенні або модернізації системи. Тому необхідно, щоб устаткування, установлюване в електричному щиті, дозволяло в будь-який момент доповнити систему будь-яким апаратом. Приміром, стаціонарна частина системи Smissline TP виготовляється у вигляді складальної панелі шасі з покладеними в них шинами й розніманнями для підключення від 6 до 108 модулів. Таким чином, якщо застосовувати це встаткування, на стадії проекту не потрібно продумувати, яка буде кінцева комбінація пристроїв, потрібні чи для них допоміжні сигнальні контакти і яке буде розподіл навантаження по фазах.

**Проблема 4. Необхідність постійно контролювати стан системи електроживлення**

Опис: Висока відказостійкість сервісів досягається тільки за допомогою додаткового моніторингу, що дозволяє запобігти перевантаженням мережі й аварії.

Контроль параметрів електромережі в режимі онлайн дає можливість обслуговуючому персоналу набагато швидше реагувати на передаварийні стани електроустаткування, вчасно запобігати відмови елементів системи й прогнозувати можливі проблеми електропостачання в цілому. При побудові більших і складних систем ЦОД моніторинг стає критично важливим, оскільки дозволяє звести до мінімуму вплив людського фактора, а також зменшити необхідна кількість оперативно-технічного персоналу й забезпечити найбільшу

оперативність у прийнятті рішень. Застосування системи контролю як мінімум бажано, а у випадку складних систем – життєво необхідно.

Рішення: Застосування спеціальних систем виміру й контролю параметрів електричної мережі. В 2015 році один з найбільших операторів мобільного зв'язку провів тестування системи виміру струмів ліній, що відходять, для вторинного розподілу енергії в одному зі своїх ЦОДів. Ґрунтуючись на результатах випробувань, був зроблений вивід про можливість застосування CMS від АББ для підвищення відказостійкості й енергоефективності систем енергопостачання IT устаткування. Керуючий модуль системи виміру струмів CMS підтримує протокол Modbus RTU, за допомогою якого передаються дані в системи диспетчеризації й управління. Мініатюрні датчики системи виміру струмів сумісні з усіма компонентами серій Smissline TP і System Pro. Для їхнього з'єднання із блоком управління використовується єдиний шлейф із цифровим інтерфейсом». За словами фахівця, важливою особливістю модульної системи виміру струмів CMS є те, що вимір струмів виробляється безпосередньо в ланцюгах споживачів. Таке рішення дає можливість одержувати інформацію навіть про незначні зміни параметрів електричної мережі й прогнозувати ймовірні поломки блоків живлення встаткування або запобігати спрацьовування пристроїв, що захищають від перевантажень.

Згідно даним незалежної організації по дослідженню ЦОД Uptime Institute®[4] за 2024 рік, в Україні функціонує лише декілька дата-центрів, сертифікованих по класі Tier 3, найбільше кількість ЦОД відноситься до рівнів Tier 1 і Tier 2. Однак варто розуміти, що тільки високий рівень надійності дата-центрів є гарантом успішної й безперебійної роботи інформаційних систем підприємства. Можна взяти приклад з європейців, які не скупляться на застосування інновацій у бізнесі й у підсумку тільки виграють, адже витрати на встаткування непорівнянні з фінансовими втратами у випадку простою систем.

### 3.2 Розробка структурної схеми

Такі природні явища, як цунамі, сильні землетруси, виверження вулканів або екстремальні повені, трапляються рідко, але й великі пожежі, ушкодження ліній або збої в подачі електроенергії можуть повністю паралізувати роботу центрів обробки даних. Щоб IT-системи продовжували функціонувати навіть у критичних умовах, підприємства роблять ставку на так звані метрокластери, або розтягнуті кластери, вузли яких розподілені між двома або більше площадками.

Висока доступність завжди досягалася шляхом забезпечення надмірності – це актуально й у випадку підготовки до екстремальних ситуацій, коли весь ЦОД необхідно захистити від збоїв у подачі електроенергії або від природних катастроф. Якщо один із ЦОД виходить із ладу, територіально рознесений кластер автоматично й без переривання робочих процесів перемикається на другий, а при необхідності й на третій центр обробки даних. По суті, це не що інше, як локальний кластер, рознесений між двома або трьома площадками, з локально дзеркальованою системою зберігання.

Відповідно до концепції територіально рознесених кластерів на кожній площадці повинен бути окремий рівень зберігання, що, у свою чергу, відповідає принципу забезпечення високої доступності, тобто являє собою кластер із двома вузлами (Node). Цей кластер надає дисковий простір для сервісних вузлів (Service Node). Останні дзеркалюють наявні дані між двома площадками, а разом чотири вузли формують один територіально розподілений чотирьохвузловий кластер.

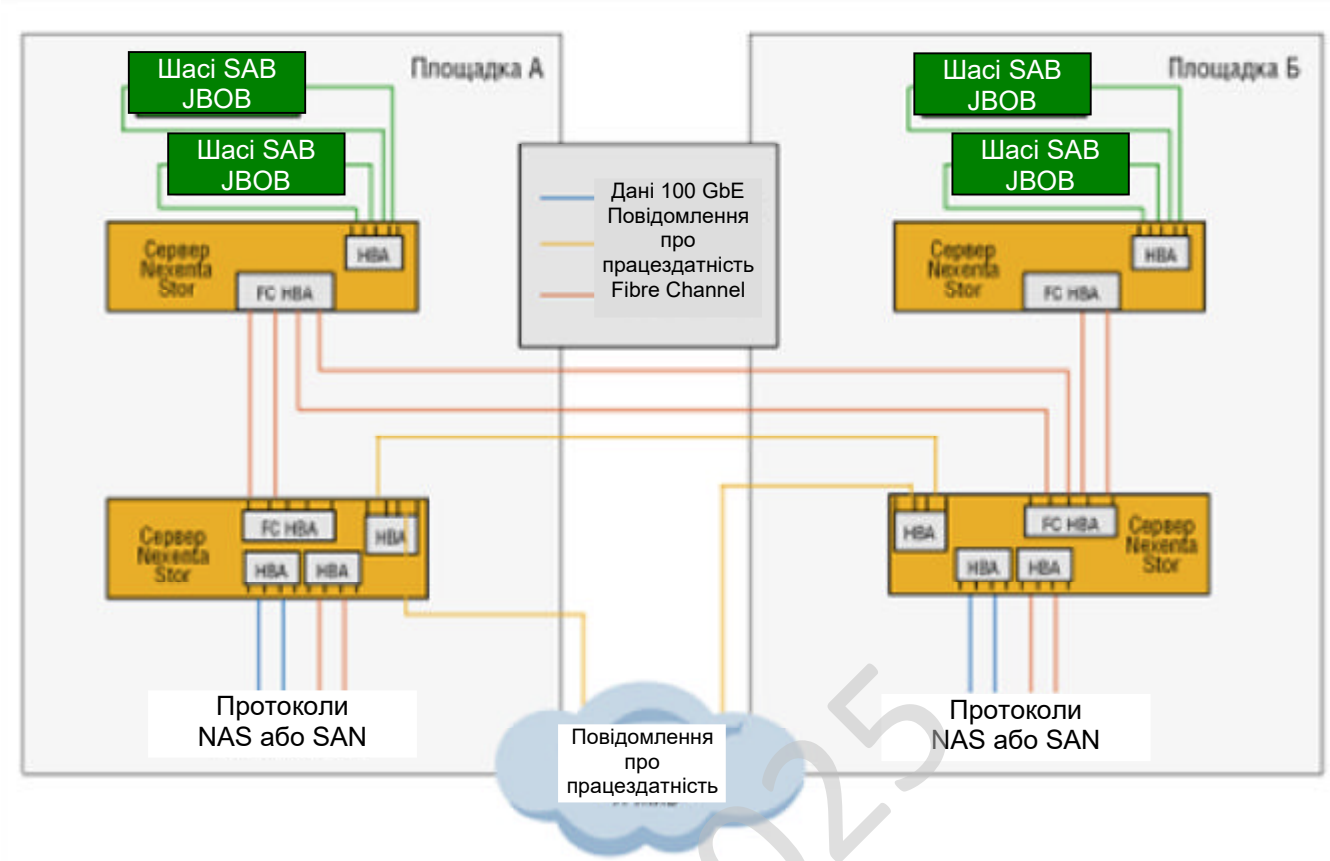


Рисунок 3.1 – Структурна схема системи

Територіально розподілені кластери можуть бути організовані таким чином, що в них не залишиться критичних крапок відмови (Single Point of Failure). Таким чином, збій у роботі апаратного забезпечення – неважливо, на якому рівні – не зажадає перемикатися між площадками вручну: велика перевага таких рішень полягає в тому, що при виникненні проблеми перемикання відбувається зовсім прозоро й без якого-небудь втручання адміністратора. Якби при цьому використовувалася тільки технологія асинхронної реплікації даних, то рішення про вживання аварійних заходів однаково довелося б приймати людині, що привело б до значних затримок. Крім того, треба була б наявність плану дій на випадок надзвичайної ситуації, у якому чітко вказувалося б, як і коли необхідно здійснювати перемикання. Автоматизація цього процесу гарантує безперервну роботу всіх додатків.

Крім захисту дані підприємства, територіальна розподіленість кластерів має ще одна важлива перевага: метрокластери не потрібно зупиняти, щоб обновити їх апаратне або програмне забезпечення. До того ж вони досить прості в реалізації й наступній експлуатації. Правда, з'єднання між площадками повинні мати дуже низькі показники затримок (Latency), тому що більші затримки негативно позначаються на продуктивності всієї системи. А оскільки зі збільшенням відстані затримки збільшуються, відстань між ЦОД не повинне перевищувати 50 км.

Таким чином, метрокластери вигідні тим підприємствам, які або займають дуже більшу територію, або мають філії, що відстоять один від іншого не більше ніж на 50 км. Із цієї причини в США ця концепція не одержала широкого поширення, адже в більшості випадків відстані між філіями компаній там набагато більше, а виходить, про метрокластери може бути й мови. У всіх інших випадках компанії зможуть при малих інвестиціях підняти доступність своїх систем на новий, раніше недосяжний рівень.

**Сценарії системного збою**

У кожного територіально розподіленого кластера безліч слабких місць, здатних паралізувати роботу системи. Тому головне завдання полягає в тому, щоб для кожного з можливих випадків надати автоматичні резервні рішення, що дозволяють запобігти збоям у роботі додатків. Розглянемо сім сценаріїв відмови систем і можливих наслідків на прикладі метрокластера:

– Відмова жорсткого диска. У цьому випадку звичайно ніяких негативних наслідків для подальшої роботи систем не буває. Адміністратор може замінити ушкоджений диск в «гарячому» режимі, після чого дані знову автоматично синхронізуються.

– Вихід з ладу важливих компонентів дискових полиць (Disk Shelves). При відмові кабелю SAS (Serial Attached SCSI), адаптера головної шини SAS-HBA (Host Bus Adapter) або розширника SAS (SAS Expander) технологія множинного доступу (Multi-Pathing) у вузлах зберігання (Storage Node) забезпечить

безперервну роботу всіх сервісів. І в цьому випадку адміністратор може провести оперативну заміну несправних елементів.

– Збій у роботі всієї дискової полиці. Масиви жорстких дисків RAID-Z2 розподіляються по системах із простим послідовним підключенням дисків (Just a Bunch of Discs, JBOD) таким чином, що навіть повна відмова однієї JBOD-системи можна пережити без втрат. Коли така система відновить роботу, синхронізовані будуть тільки ті дані, які перетерпіли зміни до цього моменту. Таким чином, всі сервіси продовжать функціонувати без простоїв і значних спадів продуктивності не буде.

– Порушення в роботі вузла зберігання. При відмові всього сервера вузла зберігання його обов'язку протягом декількох секунд переходять до другого сервера, розташованого тій же самій площадці. короткочасне переривання, Що Відбувається в цьому випадку, потоку уведення-виводу даних буде помічено вищестоящими сервісними вузлами, але не позначиться на роботі додатків, оскільки в кожний момент часу виробляється дзеркалювання площадки.

– Проблеми в роботі комутатора, кабелю або HBA-адаптера Fibre Channel між вузлами зберігання й вищестоящих сервісних вузлів. Технологія множинного доступу на сервісних вузлах дозволяє впоратися й із цим сценарієм. Резервного перемикання на другий ЦОД не буде потрібно, і кінцевий користувач не відчує ніякого зниження продуктивності додатків.

– Непрацездатність сервісного вузла. У випадку відмови всього сервісного вузла відбувається короткочасне – тривалістю в кілька секунд – переривання потоків уведення-виводу даних до додатків і від них. Час перемикання визначається кількістю використовуваних сервісів, таких як NFS Share, CIFS Share або iSCSI Target, і не залежить від обсягів даних. Одна з особливостей технології ZFS, що відрізняє її від інших файлових систем і систем зберігання, полягає в тому, що їй ніколи не потрібне проведення повної перевірки файлової системи. Для серверів додатків це перемикання прозоро; а якщо застосовується Fibre Channel, їм необхідно одержати в операційної системи драйвер Multi-Path з

підтримкою асиметричного доступу до логічних елементів (Asymmetric Logical Unit Access, ALUA), що в багатьох випадках є стандартною функцією. При цьому кластери настроєні таким чином, що у випадку збоїв сервіси спочатку переносяться на сусідні вузли й необхідність у перемиканні на територіально віддалений вузол виникає, тільки якщо робота філії порушується повністю.

– Неприступність всієї площадки. У самому гіршому варіанті можливий збій у роботі філії в цілому. Тільки в цій ситуації територіально рознесений кластер використовує надмірність на рівні ЦОД для подолання збоїти й системи, що перебувають на другій площадці, беруть на себе підтримку всіх сервісів. Таким чином, сервери додатків зберігають доступ до всіх служб, нехай і з половиною сервісних вузлів, тобто з обмеженою продуктивністю. Оскільки при такому сценарії дзеркалювання, зчитування й запис даних між територіально рознесеними філіями не виробляються, тривалість затримок скорочується. При роботі, приміром, з базами даних їхня продуктивність нерідко навіть поліпшується. Коли площадка, на якій відбувся збій, знову ввійде в робочий режим, здійснювати зворотне дзеркалювання всіх файлів не прийдеться. Передати буде потрібно тільки ті дані, які були змінені за час простою, тому після усунення локальних проблем потерпілий ЦОД зможе дуже швидко повернутися до нормальної роботи.

### 3.3 Розробка функціональної схеми

На рисунку 3.2 зображена функціональна схема системи. З неї бачимо, що розроблена система підвищення надійності функціонування ЦОД з наступних основних функціональних блоків:

– сам носій інформації;

– кодер циклічного коду, який використовується, коли відбувається запис інформації на носій інформації, при цьому необхідно враховувати, що об’єм інформації, яка записується на носій інформації, повинна бути меншою, ніж

об’єм носія інформації, в зв’язку, з тим, що коди циклічного коду відносяться до кодів з надмірністю, за рахунок якої й відбувається кодування;

– декодер циклічного коду, який використовується при читанні даних с відповідного носія інформації;

– файли для перешкодостійкого збереження;

– відновлені файли.

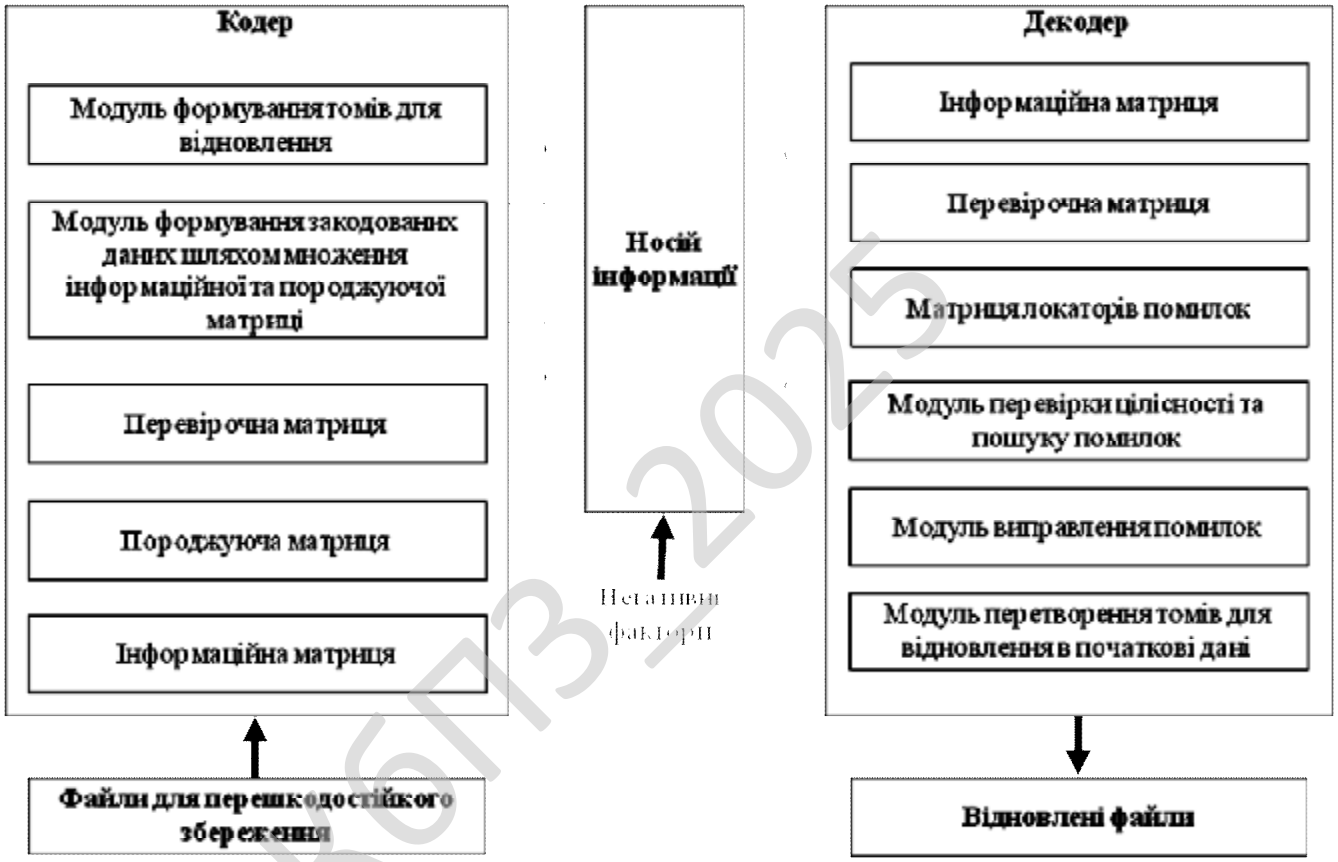


Рисунок 3.2 – Функціональна схема системи

Розглянемо більш детально перераховані функціональні блоки кодування та декодування за допомогою циклічного кодування.

Блок кодування складається з наступних підблоків:

– Модуль формування томів для відновлення.

– Модуль формування закодованих даних шляхом множення інформаційної та породжуючої матриці.

– Перевірочна матриця.

– Породжуюча матриця.

– Інформаційна матриця.

Блок декодування складається з наступних підблоків:

– Інформаційна матриця.

– Перевірочна матриця.

– Матриця локаторів помилок.

– Модуль перевірки цілісності та пошуку помилок.

– Модуль виправлення помилок.

– Модуль перетворення томів для відновлення в початкові дані.

Коди циклічного коду базуються на спеціальному розділі математики – полях Галуа (GF) або кінцевих полях. Арифметичні дії (+, -, х, / і т.д.) над елементами кінцевого поля дають результат, що також є елементом цього поля. Кодер та декодер циклічного коду повинні вміти виконувати ці арифметичні операції. Ці операції для своєї реалізації вимагають спеціального устаткування або спеціалізованого програмного забезпечення.

Кодове слово циклічного коду формується із залученням спеціального полінома. Всі коректні кодові слова повинні ділитися без залишку на ці утворюючі поліноми. Загальна форма утворюючого полінома має вигляд:

$$g(x) = (x\text{-}a^{i})(x\text{-}ai^{+1})\ldots(x\text{-}a^{i+2t}),$$

а кодове слово формується за допомогою операції:

$$c(x) = g(x)i(x),$$

де – $g(x)$ є утворюючим поліномом;

– $i(x)$ являє собою інформаційний блок;

– $c(x)$ – кодове слово, що називається простим елементом поля.

$2t$ символів парності в кодовому слові циклічного коду визначаються з наступного співвідношення:

$$p(x) = i(x)\cdot x^{n-k} \bmod g(x).$$

Перейдемо до розгляду іншого функціонального блоку – декодеру циклічного коду.

Декодер працює наступним чином.

Введемо позначення:

– $r(x)$ – Отримане кодове слово.

– $S_i$ – Синдроми.

– $L(x)$ – Поліном локації помилок.

– $X_i$ – Положення помилок.

– $Y_i$ – Значення помилок.

– $c(x)$ – Відновлене кодове слово.

– $v$ – Число помилок.

Отримане кодове слово r(x) являє собою вихідне (передане) кодове слово c(x) плюс помилки: $r(x) = c(x) + e(x)$.

Декодер циклічного коду намагається визначити позицію й значення помилки для числа $t$ помилок (або $2t$ втрат) і виправити помилки й втрати.

**Обчислення синдрому**

Обчислення синдрому схоже на обчислення парності. Кодове слово циклічного коду має $2t$ **синдромів**, це залежить тільки від помилок (а не переданих кодових слів). Синдроми можуть бути обчислені шляхом підстановки $2t$ коріння утворюючого полінома $g(x)$ в $r(x)$.

**Знаходження позицій символьних помилок**

Це робиться шляхом рішення системи рівнянь із t невідомими. Існує кілька швидких алгоритмів для рішення цього завдання. Ці алгоритми використовують особливості структури матриці кодів циклічного коду й сильно скорочують необхідну обчислювальну потужність. Робиться це у два етапи:

1. Визначення полінома локації помилок

Це може бути зроблене за допомогою алгоритму Berlekamp-Massey або алгоритму Евкліда. Алгоритм Евкліда використовується частіше на практиці, тому що його легше реалізувати, однак, алгоритм Berlekamp-Massey дозволяє

одержати більш ефективну реалізацію встаткування й програм.

2. Знаходження кореня цього полінома. Це робиться із залученням алгоритму пошуку Chien.

**Знаходження значень символьних помилок**

Тут також потрібно вирішити систему рівнянь із t невідомими. Для рішення використовується швидкий алгоритм Forney.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

**Опис циклічного коду, який використовується для підвищення надійності функціонування ЦОД**

Опишемо процес кодування та декодування інформації за допомогою циклічного кодування.

Циклічним кодом називається лінійний блоковий ($n$,$k$)-код, що характеризується властивістю циклічності, тобто зрушення вліво на один крок будь-якого дозволеного кодового слова дає також дозволене кодове слово, що належить цьому ж коду й у якого, множина кодових слів представляється сукупністю багаточленів ступеня ($n$-$1$) і менш, що діляться на деякий багаточлен $g(x)$ ступеня $r = n$-$k$, що є співмножником двочлена $x^n+1$.

Багаточлен $g(x)$ називається породжуючим.

Як треба з визначення, у циклічному коді кодові слова представляються у вигляді багаточленів:

$$a(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1x^1 + a_0x^0, \qquad (3.1)$$

де $n$ – довжина коду;

$a_i$ – коефіцієнти з поля $GF(q)$.

Якщо код побудований над полем $GF(2)$, то коефіцієнти приймають значення 0 або 1 і код називається двійковим.

Наприклад, якщо код побудований над полем $GF(q)=GF(2^3)$, що є розширенням $GF(2)$ по модулі багаточлена, що не *приводиться,* $f(z)=z^3+z+1$, а елементи цього поля мають вигляд, представлений у таблиці 3.1.

То коефіцієнти $a_i(x)$ приймають значення елементів цього поля й тому вони самі відображаються у вигляді багаточленів наступного виду:

$$a_i(z) = a_{m-1} \cdot z^{m-1} + a_{m-2} \cdot z^{m-2} + \ldots + a_1 \cdot z^1 + a_0 \cdot z^0, \qquad (3.2)$$

де m – ступінь багаточлена, по якому отримане розширення поля *GF*(2);

$\alpha_i$ – коефіцієнти, що приймають значення елементів *GF*(2), тобто 0 і 1. Такий код називається *q*-ним.

Таблиця 3.1 – Елементи поля *GF*(2)

| | | | | | |
|---|---|---|---|---|---|
| 0 | 000 | 0 | $\alpha^3$ | 011 | Z+1 |
| $\alpha^0$ | 001 | 1 | $\alpha^4$ | 110 | $Z^2$+Z |
| $\alpha^1$ | 010 | Z | $\alpha^5$ | 111 | $Z^2$+Z+1 |
| $\alpha^2$ | 100 | $Z^2$ | $\alpha^6$ | 101 | $Z^2$+1 |

Довжина циклічного коду називається примітивної й сам код називається примітивним, якщо його довжина $n = q^{m-1}$ на *GF*(*q*).

Якщо довжина коду менше довжини примітивного коду, то код називається вкороченим або непримітивним.

Як треба з визначення загальна властивість кодових слів циклічного коду – це їхня подільність без остачі на деякий багаточлен *g*(*x*), називаний породжуючим.

Результатом ділення двочлена $x^n+1$ на багаточлен *g*(*x*) є перевірочний багаточлен *h*(*x*).

**Матричне завдання кодів**

Циклічний код може бути заданий породжуючий й перевірочною матрицями. Для їхньої побудови досить знати породжуючий *g*(*x*) і перевірочний *h*(*x*) багаточлени.

Для несистематичного циклічного коду матриці будуються циклічним зрушенням породжуючого й перевірочного багаточленів, тобто шляхом їхнього множення на *x*:

$$G_{(n,k)} = \begin{vmatrix} g(x) \\ x \cdot g(x) \\ x^2 \cdot g(x) \\ \bullet\bullet\bullet \\ x^{k-1} \cdot g(x) \end{vmatrix},$$

та:

$$H_{(n,k)} = \begin{vmatrix} h(x) \\ x \cdot h(x) \\ x^2 \cdot h(x) \\ \bullet\bullet\bullet \\ x^{r-1} \cdot h(x) \end{vmatrix}.$$

При побудові матриці $H_{(n,k)}$ старший коефіцієнт багаточлена $h(x)$ розташовується праворуч.

Для систематичного циклічного коду матриця $G_{(n,k)}$ визначається з вираження:

$$G_{(n,k)} = \left| I_k, R_{k,r} \right|, \tag{3.3}$$

де $I_k$ – одинична матриця;

$R_{k,r}$ – прямокутна матриця.

Рядки матриці $R_{k,r}$ визначаються з виражень:

$$r_i(x) = R_{g(x)} \left| a_i(x) \cdot x^r \right|, \tag{3.4}$$

або:

$$r_i(x) = R_{g(x)} \left| x^{n-i} \right|, \tag{3.5}$$

де $a_i(x)$ – значення і-*того* рядка матриці $I_k$;

$i$ – номер рядка матриці $R_{k,r}$.

Використовуючи вираження:

$$r_i(x) = R_{g(x)} \left| x^{n-i} \right|, \tag{3.6}$$

одержимо той же результат.

Рядки матриці $G_{(n,k)}$ можна визначити безпосередньо з вираження:

$$r_i(x) = a_i(x) \cdot x^r + r_i(x), \tag{3.7}$$

де:

$$r_i(x) = R_{g(x)}\left[a_i(x) \cdot x^r\right]. \tag{3.8}$$

Перевірочна матриця в систематичному виді будується на основі матриці $G_{(n,k)}$, а саме:

$$H_{(n,k)} = \left[R^T_{k,r}, I_r\right], \tag{3.9}$$

де $I_r$ – одинична матриця;

$R^T_{k,r}$ – матриця з $G_{(n,k)}$ у транспонованому виді.

Одна з основних задач, що коштують перед розроблювачами пристроїв захисту від помилок при передачі дискретних повідомлень по каналах зв'язку є вибір багаточлена, породжуючий, $g(x)$ для побудови циклічного коду, що забезпечує необхідну мінімальну кодову відстань для гарантійного виявлення й виправлення t-кратних помилок.

Існують спеціальні таблиці на вибір $g(x)$ залежно від пропонованих вимог до коригувальних можливостей коду. Однак у кожного циклічного коду є свої особливості формування $g(x)$. Тому при вивченні конкретних циклічних кодів будуть розглядатися відповідні способи побудови $g(x)$.

Задача кодування полягає у формуванні по інформаційних словах $a(x)$ кодових слів $(x)$ циклічного $(n,k)$-коду, що по своїй структурі може бути несистематичним і систематичним.

Формування кодових слів несистематичного коду полягає в множенні багаточлена $a(x)$, що відображає інформаційну послідовність довжини $k$, на породжуючий багаточлен, тобто $(x)=a(x)\times g(x)$. Формування кодових слів систематичного коду полягає в перетворенні інформаційної послідовності $a(x)$ відповідно до вираження $(x)=a(x)\oplus x^r+r(x)$.

Перевірочна послідовність $r(x)$ визначається двома способами:

– при використанні "класичного" способу кодування:

$$r(x) = R_{g(x)}\left[a(x)\cdot x^{r}\right]; \qquad (3.10)$$

– при використанні способу кодування, рекомендованого МККТТ:

$$r(x) = \overline{R}_{g(x)}\left[a(x)\cdot x^{r} + x(1)^{r-1}\cdot x^{k}\right], \qquad (3.11)$$

де $x(1)^{r-1}$ – одиничний багаточлен ступеня ($r$-$1$).

Зазначені вище математичні операції виконують кодерыи несистематичного й систематичного кодів.

**Способи декодування з виявленням помилок**

Процедура декодування циклічного коду з виявленням помилок, за аналогією із процесом кодування, використовує два способи:

– При кодуванні "класичним" способом декодування засноване на використанні властивості подільності без остачі кодового багаточлена ($x$) циклічного ($n$,$k$)-коду на породжуючий багаточлен $g($x). Тому алгоритм декодування містить у собі ділення прийнятого кодового слова, описуваного багаточленом $x^{n}(x)$ на $g(x)$, обчислення й аналіз остачі $r(x)$. Якщо $r(x)$=0, то прийняте кодове слово вважається неспотвореним. Якщо $r(x)$<>0, то прийняте кодове слово стирається й формується сигнал "помилка".

– При кодуванні способом МККТТ декодування засноване на властивості одержання певної контрольної остачі $R_0(x)$ при діленні прийнятого кодового багаточлена ($x$) на породжуючий багаточлен. Тому, якщо отриманий при діленні остача $\overline{r(x)} = R_0(x)$, то прийняте кодове слово вважається неспотвореним. Якщо остача $r(x) \neq R_0(x)$, то прийняте кодове слово стирається й формується сигнал "помилка". Значення контрольної остачі визначається з вираження:

$$R_0(x) = R_{g(x)}\left[x(1)^{r-1}\cdot x^{k}\right]. \qquad (3.12)$$

**Способи декодування з виправленням помилок і схемна реалізація декодувальних пристроїв**

Декодування циклічного коду в режимі виправлення помилок можна здійснювати різними способами. Нижче викладаються два способи, що є найбільш простими.

В основу першого способу покладене використання таблиці синдромів (декодування), у якій кожному багаточлену або зразку помилок $e_i(x)$, відповідає певний синдром $S_i(x)$, що представляє остачу від ділення прийнятого кодового слова $a_i'(x)$ й відповідного йому $e_i(x)$ на $g(x)$. Процедура декодування наступна. Прийняте кодове слово $a_i'(x)$ ділиться на $g(x)$, визначається $S_i(x)$ і відповідний йому багаточлен $e_i(x)$, а потім $a_i'(x)$ підсумується з $e_i(x)$. У результаті одержуємо виправлене кодове слово, тобто:

$$a_i(x) = a_i'(x) + e_i(x). \tag{3.13}$$

До складу декодера входять: обчислювач синдрому (ВР), два регістри зрушення *RG*1 і *RG*2, постійний запам'ятовувальний пристрій (ПЗП), що містить $\sum_{i=1}^{t_u} C_n^i$ слова довжини $n$, що відповідають багаточленам помилок $e_i(x)$.

Прийняте кодове слово $a_i'(x)$ надходить на вхід обчислювача синдрому, де здійснюється ділення його на $g(x)$ і формування $S_i(x)$, і одночасно – на вхід *RG*2, де $a_i'(x)$ накопичується. Синдром $S_i(x)$ використовується як адреса, по якому із ПЗП в регістр *RG*1 записується $e_i(x)$, що відповідає синдрому $S_i(x)$. Перераховані операції завершуються за $n$ тактів. Протягом наступних $n$ тактів відбувається заелементне підсумовування вмісту *RG*2 і *RG*1, тобто операція:

$$a_i(x) = a_i'(x) + e_i(x), \tag{3.14}$$

і виправлення помилок.

В основі другого способу виправлення помилок, що дозволяє значно скоротити об'єм використовуваних табличних синдромів і істотно спростити схему декодера, лежать наступні положення:

1. Синдром $S_i(x)$, що відповідає прийнятому кодовому слову дорівнює остачі від ділення $a_i'(x)$ на $g(x)$, а також остачі від ділення відповідного багаточлена помилок $ei(x)$ на $g(x)$, тобто:

$$S_i(x) = R_{g(x)}[v_i'(x)] = R_{g(x)}[e_i(x)]. \tag{3.15}$$

2. Якщо $S_i(x)$ відповідає $a_i'(x)$ й $e_i(x)$, то $x \in S_i(x)$ є синдромом, що відповідає:

$$R_{g(x)}\left[x \cdot S_i(x)\right] = R_{g(x)}\left[x \cdot a_i'(x) \bmod (x^n - 1)\right] = \\ = R_{g(x)}\left[x \cdot e_i(x) \bmod (x^n - 1)\right], \quad (3.16)$$

і:

$$x \cdot a_i'(x) \bmod (x^n - 1), \quad (3.17)$$

або:

$$x \cdot e_i(x) \bmod (x^n - 1). \quad (3.18)$$

3. При виправленні помилок використовуються синдроми зразків помилок тільки з ненульовим коефіцієнтом у старшому розряді.

Тому при реалізації цього способу множина всіх зразків помилок розбивається на класи еквівалентності. Кожний клас представляє циклічне зрушення одного зразка помилок, а синдром цього класу відповідає зразку помилок з ненульовим старшим розрядом. Якщо обчислений синдром належить одному із класів еквівалентності зразків помилок, що виправляються, то старший символ кодового слова виправляється. Потім прийняте слово й синдром циклічно зрушується, а процес знаходження в попередній по старшинству позиції повторюється.

Для виправлення помилок, що належать даному класу еквівалентності, потрібно зробити $n$ циклічних зрушень.

Найпростішим є декодер Меггітта. До складу декодера входять: обчислювач синдрому, що здійснює ділення кодового слова $a_i'(x)$ на $g(x)$ і формування відповідного синдрому; блок декодерів (ДК), що настроєний на синдроми всіх зразків помилок, що виправляються, з ненульовими старшими розрядами; регістр зрушення *RG*.

При надходженні на вхід схеми кодового слова $a_i'(x)$ його символи заповнюють регістр *RG*, а в обчислювачі формується відповідний синдром $S_i(x)$. Обчислений синдром рівняється з усіма табличними синдромами, закладеними в

схему блоку ДК, і у випадку збігу з одним з них на його виході формується сигнал, що виправляє помилковий символ, що перебуває в старшому розряді регістра. Після цього вміст обчислювача й *RG* циклічно зрушується на один крок. Це зрушення реалізує операції $R_{g(x)}[x \cdot S_i(x)]$ й $x \cdot v'(x) \bmod(x^n - 1)$. Якщо новий синдром збігається з одним з табличних синдромів, то це означає, що відбулася помилка в другому по старшинству символі кодового слова, що, перейшовши в старший розряд *RG*, виправляється. Потім виробляється нове циклічне зрушення на одну позицію й нову перевірку на збіг синдромів. Після повторення цього процесу *n* раз в *RG* буде сформоване виправлене кодове слово. Введення зворотного зв'язка для *RG* не обов'язково, тому що в процесі виправлення помилок символи кодового слова надходять на вихід декодера.

При декодуванні циклічних кодів використовуються багаточлен помилок *e*(*x*) і синдромний багаточлен *S*(*x*).

Багаточлен помилок ступеня не більше (*n*-1) визначається з вираження:

$$e(x) = v'(x) + v(x), \tag{3.19}$$

де $v'(x)$ *и* $v(x)$ – багаточлени, що відображають відповідно прийняте (з помилкою) і передане кодові слова.

Ненульові коефіцієнти в *e*(*x*) займають позиції, які відповідають помилкам.

Синдромний багаточлен, використовуваний при декодуванні циклічного коду, визначається як остача від ділення прийнятого кодового слова на породжуючий багаточлен, тобто:

$$S_i(x) = R_{g(x)}[v'(x)], \tag{3.20}$$

або:

$$S_i(x) = R_{g(x)}[v'(x) + e_i(x)] = R_{g(x)}[e_i(x)]. \tag{3.21}$$

Отже, синдромний багаточлен залежить безпосередньо від багаточлена помилок *e*(*x*).Це положення використовується при побудові таблиці синдромів,

застосовуваної в процесі декодування. Ця таблиця містить список багаточленів помилок і список відповідних синдромів, обумовлених з вираження:

$$S_i(x) = R_{g(x)}[e_i(x)] \tag{3.22}$$

Таблиця 3.2 – Список багаточленів помилок і список відповідних синдромів

| $(x)$ | $S(x)$ |
|---|---|
| 1 | $R_{g(x)}[1]$ |
| $X$ | $R_{g(x)}[x]$ |
| $X^2$ | $R_{g(x)}[x^2]$ |
| $X+1$ | $R_{g(x)}[x+1]$ |
| $X^2+1$ | $R_{g(x)}[x^2+1]$ |

У процесі декодування по прийнятому кодовому слову обчислюється синдром, потім у таблиці перебуває відповідний багаточлен $e(x)$, підсумовування якого із прийнятим кодовим словом дає виправлене кодове слово, тобто:

$$v_i(x) = v_i'(x) + e_i(x). \tag{3.23}$$

Перераховані багаточлени $v(x), v'(x), g(x), h(x), e(x)$ и $S(x)$ можна складати, множити й ділити, використовуючи відомі правила алгебри, але із приведенням результату по mod 2, а потім по mod $x^n+1$, якщо ступінь результату перевищує ступінь $(n-1)$.

При побудові й декодуванні циклічних кодів у результаті ділення багаточленів звичайно необхідно мати не частка, а остача від ділення.

Тому рекомендується більш простий спосіб ділення, використовуючи не багаточлени, а тільки його коефіцієнти.

### 3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.3. При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі.

Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

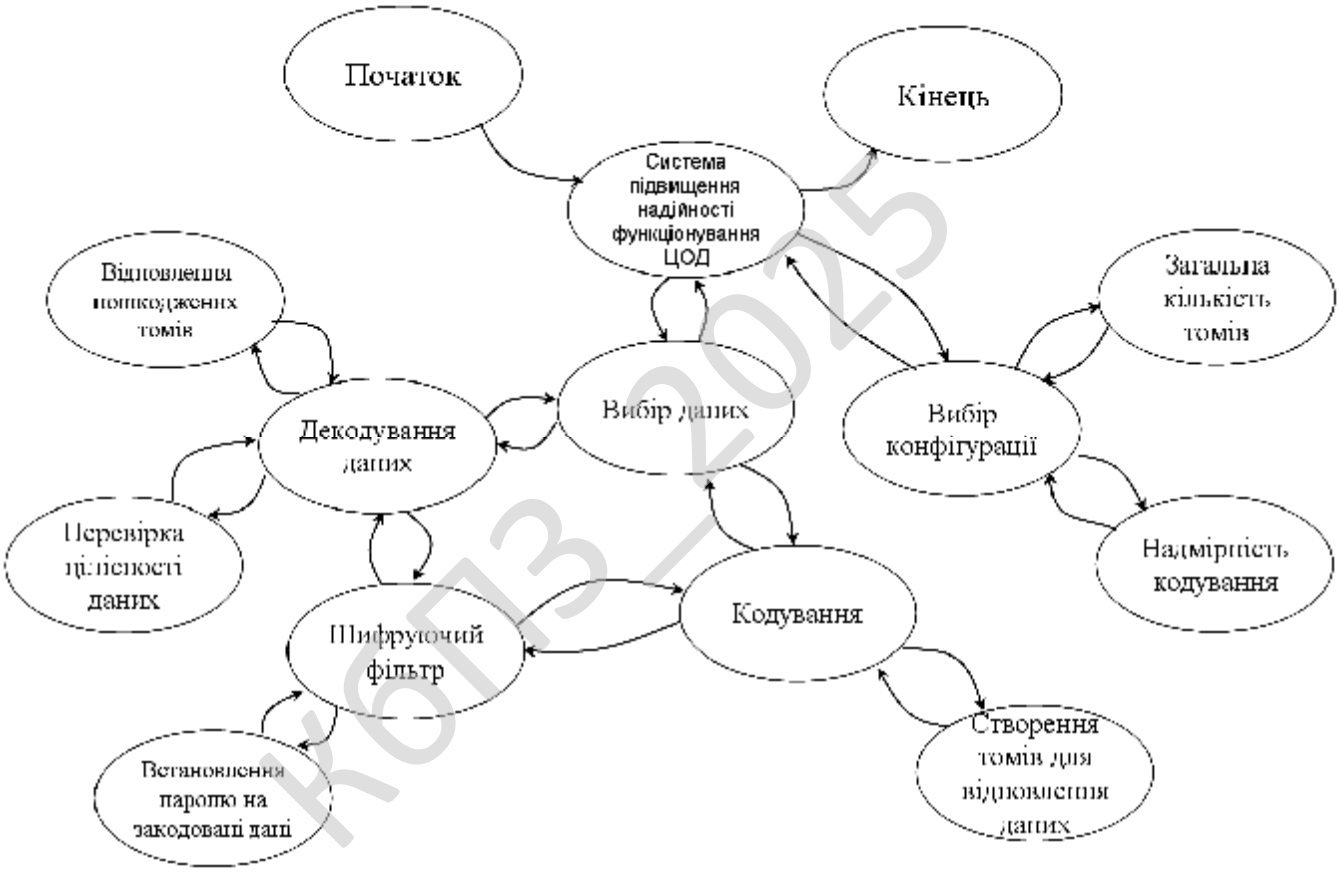


Рисунок 3.3 – Діаграма взаємодії процесів

Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування). Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи.

Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

Діаграми потоків даних містять чотири типи елементів:

– Процеси які являють собою трансформацію даних в рамках описуваної системи.

– Сховища даних (репозиторії).

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

# 4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

## 4.1 Блок-схеми та опис алгоритмів функціонування системи

Було використано підходи з використанням UML, це уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної UML-моделлю. UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація.

UML може бути застосовано на всіх етапах життєвого циклу аналізу бізнес-систем і розробки прикладних програм. Різні види діаграм які підтримуються UML, і найбагатший набір можливостей представлення певних аспектів системи робить UML універсальним засобом опису як програмних, так і ділових систем.

Діаграми дають можливість представити систему (як ділову, так і програмну) у такому вигляді, щоб її можна було легко перевести в програмний код. Основною причиною використання мови UML є спілкування розробників між собою.

Крім того, UML спеціально створювалася для оптимізації процесу розробки програмних систем, що дозволяє збільшити ефективність їх реалізації у кілька разів і помітно поліпшити якість кінцевого продукту.

UML прекрасно зарекомендувала себе в багатьох успішних програмних проектах. Засоби автоматичної генерації кодів дозволяють перетворювати моделі мовою UML у вихідний код об’єктно-орієнтованих мов програмування, що ще більш прискорює процес розробки. Практично усі CASE-засоби (програми автоматизації процесу аналізу і проектування) мають підтримку UML. Моделі розроблені в UML, дозволяють значно спростити процес кодування і направити зусилля програмістів безпосередньо на реалізацію системи.

Діаграми підвищують супроводжуваність проекту і полегшують розробку документації.

UML необхідний:

– Керівникам проектів, які керують розподілом завдань і контролем за проектом.

– Проектувальникам інформаційних систем які розробляють технічні завдання для програмістів.

– Бізнес-аналітикам, які досліджують реальну систему і здійснюють інжиніринг і реінжиніринг бізнесу компанії.

– Програмістам які реалізовують модулі інформаційної системи.

При модифікації системи об'єктний підхід дозволяє легко включати в систему нові об'єкти і виключати застарілі без істотної зміни її життєздатності. Використання побудованої моделі при модифікаціях системи дає можливість усунути небажані наслідки змін, оскільки вони не ламають структури системи, а тільки змінюють поведінку об'єктів.

Під час роботи над бакалаврською дипломною роботою було створено блок-схеми. Перед їх розглядом необхідно провести роз’яснення який саме тип блок-схем використовується.

Блок-схема це представлення задачі для її аналізу або розв'язування за допомогою спеціальних символів (геометричних образів), які позначають такі елементи, як операції, потік, дані тощо. Блок вхідних та вихідних даних прийнято

позначати паралелограмом, блок обчислень (обробки) даних – прямокутником, блок прийняття рішень – ромбом, еліпсом – початок та кінець алгоритму.

У інформаційних технологіях функціональна схема складається з функціональних блоків, які являють собою конструктивно відособлені частини (елементи або пристрої) автоматичних систем, які виконують певні функції. Функціональні блоки на схемі позначають прямокутниками, всередині яких надписують їх найменування відповідно до функцій, що виконуються. Зв'язки між функціональними блоками (внутрішні впливи) позначаються лініями зі стрілками, які вказують напрям впливів.

Функціональні схеми можуть виконуватися в укрупненому і розгорненому вигляді. У першому випадку на схемі зображають найважливіші блоки системи і зв'язки між ними.

У другому варіанті схема відображається більш детально, що полегшує її читання та ілюструє принцип роботи.

Основні елементи схем алгоритму це термінатор, процес, рішення, зумовлений процес (підпрограма), дані та з'єднувач.

Термінатор це елемент відображає вхід із зовнішнього середовища або вихід з неї (найчастіше застосування – початок і кінець програми). Всередині фігури записується відповідна дія.

Процес це виконання однієї або кількох операцій, обробка даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції.

Рішення це показує рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу. Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елементу. Якщо виходів два чи три то зазвичай кожен вихід позначається лінією, що виходить з решти вершин (бічних і нижній). Якщо виходів більше трьох, то їх слід показувати однією лінією, що виходить з вершини (частіше нижній) елемента, яка

потім розгалужується. Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи.

Зумовлений процес (підпрограма) це символ відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані.

Дані це перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи).

З'єднувач це символ відображає вихід в частину схеми і вхід з іншої частини цієї схеми. Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.

На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми.

З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ. При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

Також при розробці бакалаврської дипломної роботи було використано наступні підходи UML: діаграма діяльності (діаграми поведінки типу); діаграма прецедентів (діаграми поведінки типу); Діаграма класів; Діаграма компонент.

Діаграма діяльності. Це візуальне представлення графу діяльностей. Граф діяльностей є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходи відбуваються по завершенню дій.

Дія є фундаментальною одиницею визначення поведінки в специфікації. Дія отримує множину вхідних сигналів, та перетворює їх на множину вихідних сигналів.

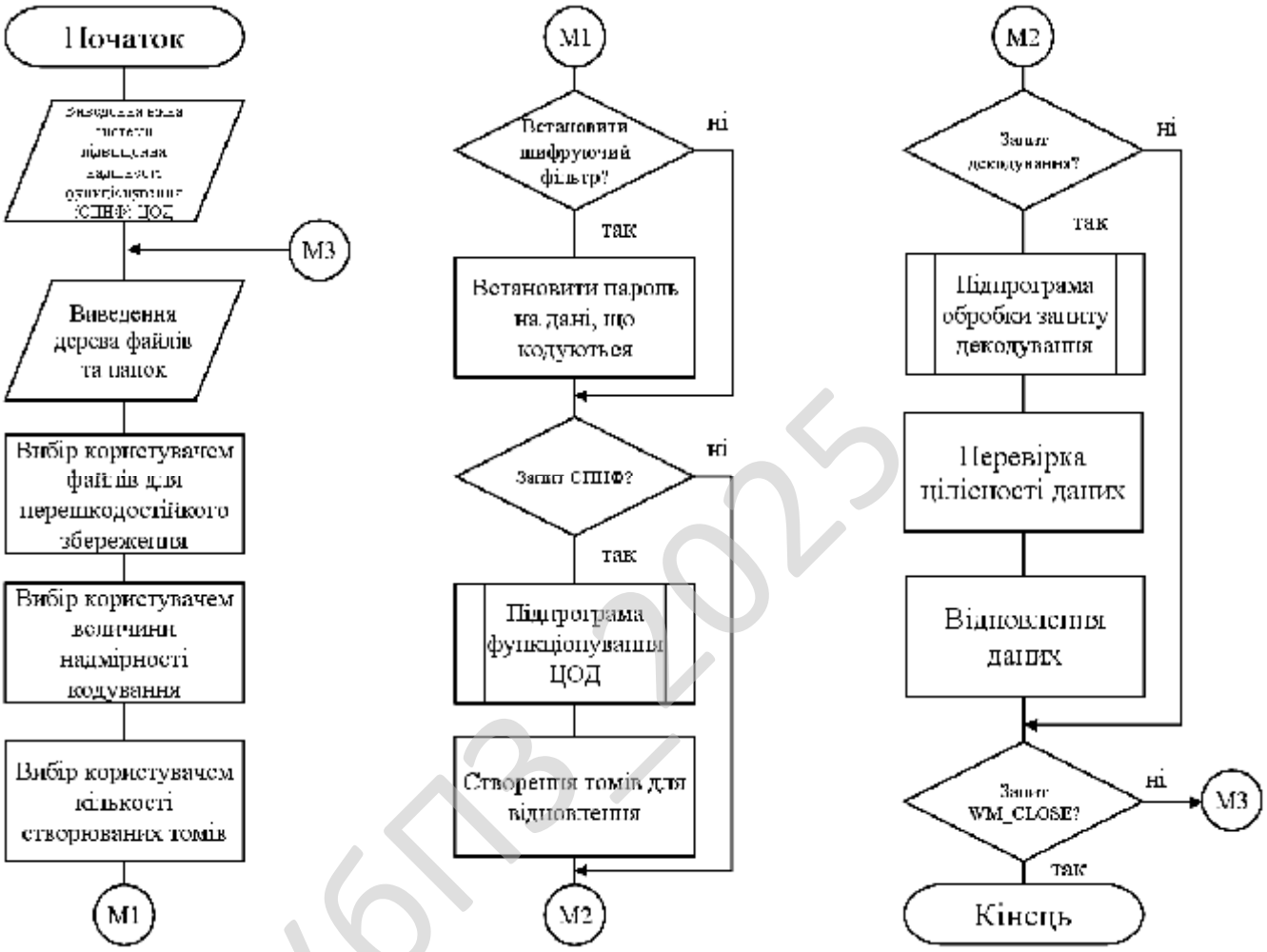


Рисунок 4.1 – Блок-схема основної програми

Одна із цих множин, або обидві водночас, можуть бути порожніми. Виконання дії відповідає виконанню окремої дії. Подібно до цього, виконання діяльності є виконанням окремої діяльності, буквально, включно із виконанням тих дій, що містяться в діяльності. Кожна дія в діяльності може виконуватись один, два, або більше разів під час одного виконання діяльності. Щонайменше, дії мають отримувати дані, перетворювати їх та тестувати, деякі дії можуть вимагати певної послідовності.

Специфікація діяльності (на вищих рівнях сумісності) може дозволяти виконання декількох (логічних) потоків, та існування механізмів синхронізації для гарантування виконання дій у правильному порядку.

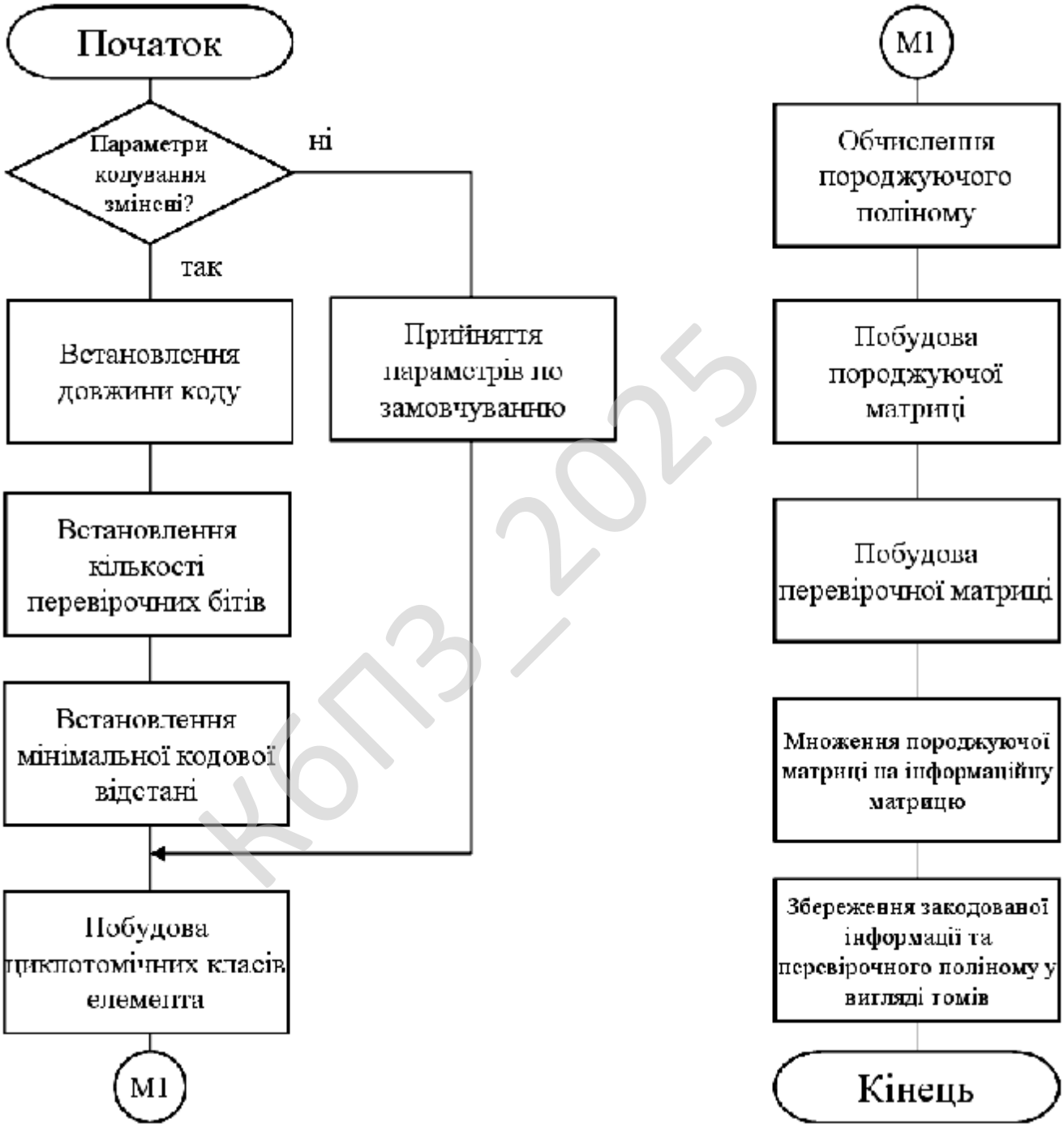


Рисунок 4.2 – Блок-схема роботи підпрограми

Діаграма прецедентів це діаграма, на якій зображено відношення між акторами та прецедентами в системі. Також, перекладається як діаграма варіантів використання.

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання (use case) використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система при діалозі з актором.

При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою.

У мові UML є кілька стандартних видів відношень між акторами і варіантами використання:

– асоціації (association relationship);

– включення (include relationship);

– розширення (extend relationship);

– узагальнення (generalization relationship).

При цьому загальні властивості варіантів використання можуть бути представлені трьома різними способами, а саме – за допомогою відношень включення, розширення і узагальнення.

Відношення асоціації – одне з фундаментальних понять у мові UML і в тій чи іншій мірі використовується при побудові всіх графічних моделей систем у формі канонічних діаграм.

Включення (include) у мові UML – це різновид відношення залежності між базовим варіантом використання і його спеціальним випадком. При цьому відношенням залежності (dependency) є таке відношення між двома елементами моделі, при якому зміна одного елемента (незалежного) приводить до зміни іншого елемента (залежного).

Відношення розширення (extend) визначає взаємозв'язок базового варіанта використання з іншим варіантом використання, функціональна поведінка якого задіюється базовим не завжди, а тільки при виконанні додаткових умов.

Діаграма класів це статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення.

Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм.

Діаграма класів (class diagram) служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини.

В UML існують наступні типи зв'язків які використовуються у діаграмі класів: Асоціації; Агрегація; Композиція.

Асоціації це якщо між двома класами визначена асоціація, то можна переміщатися від об'єктів одного класу до об'єктів іншого. Цілком припустимі випадки, коли обидва кінці асоціації відносяться до одного і того ж класу. Це означає, що з об'єктом деякого класу дозволено зв'язати інші об'єкти з того ж класу. Асоціація, що зв'язує два класи, називається бінарної. Можна, хоча це рідко буває необхідним, створювати асоціації, що зв'язують відразу кілька класів. Графічно асоціація зображується у вигляді лінії, що з'єднує клас сам з собою або з іншими класами.

Асоціації може бути присвоєно ім'я, яке описує природу відносини. Зазвичай ім'я асоціації не вказується, якщо тільки ви не хочете явно задати для неї рольові імена або у вашій моделі настільки багато асоціацій, що виникає необхідність посилатися на них і відрізняти один від одного. Ім'я буде особливо корисним, якщо між одними і тими ж класами існує кілька різних асоціацій.

Клас, що бере участь в асоціації, грає в ній деяку роль. По суті, це "обличчя", яким клас, що знаходиться на одній стороні асоціації, звернений до класу з іншого її боку. Можна явно позначити роль, яку клас грає в асоціації.

Часто при моделюванні буває важливо вказати, скільки об'єктів може бути пов'язано допомогою одного примірника асоціації. Це число називається кратністю (Multiplicity) ролі асоціації та записується або як вираз, значенням якого є діапазон значень, або в явному вигляді.

Вказуючи кратність на одному кінці асоціації, ви тим самим говорите, що на цьому кінці саме стільки об'єктів повинно відповідати кожному об'єкту на протилежному кінці. Кратність можна задати рівною одиниці (1), можна вказати діапазон: "нуль або одиниця" (0..1), "багато" (0 .. *), "одиниця або більше" (1 .. *). Дозволяється також вказувати певне число (наприклад, 3). За допомогою списку можна задати і більш складні кратності, наприклад 0. . 1, 3..4, 6 .. *, що означає "будь-яке число об'єктів, крім 2 і 5".

Агрегація це проста асоціація між двома класами відображає структурний відношення між рівноправними сутностями, коли обидва класу знаходяться на одному концептуальному рівні і ні один не є більш важливим, ніж інший. Але іноді доводиться моделювати відношення типу «частина/ціле», в якому один з класів має більш високий ранг (ціле) і складається з декількох менших за рангом (частин).

Ставлення такого типу називають агрегацією; воно зараховане до відносин типу «має» (з урахуванням того, що об'єкт-ціле має кілька об'єктів-частин). Агрегація є окремим випадком асоціації і зображується у вигляді простої асоціації

з незафарбованим ромбом з боку «цілого». Графічно агрегація представляється порожнім ромбом на блоці класу, і лінією, яка від цього ромба до міститься класу.

Композиція це більш суворий варіант агрегації. Відома також як агрегація за значенням.

Композиція має жорстку залежність часу існування екземплярів класу контейнера та примірників містяться класів. Якщо контейнер буде знищений, то весь його вміст буде також знищено. Графічно представляється як і агрегація, але з зафарбовані ромбиком.

Діаграма компонент в UML це діаграма, на якій відображаються компоненти, залежності та зв'язки між ними.

Діаграма компонент відображає залежності між компонентами програмного забезпечення, включаючи компоненти вихідних кодів, бінарні компоненти, та компоненти, що можуть виконуватись.

Модуль програмного забезпечення може бути представлено в якості компоненти. Деякі компоненти існують під час компіляції, деякі – під час компонування, а деякі під час роботи програми.

Діаграма компонент відображає лише структурні характеристики, для відображення окремих екземплярів компонент слід використовувати діаграму розгортування.

Компоненти об'єднуються разом використовуючи структурні зв'язки (assembly connector) щоб об'єднати інтерфейси двох компонент. Це ілюструє зв'язок типу «клієнт-сервер».

Структурна взаємодія – «зв'язок двох компонент, який передбачає, що один з них надає послуги, потрібні іншому компоненту».

При використанні діаграми компонент щоб показати внутрішню структуру компонента, клієнтські та серверні інтерфейси можуть утворювати пряме з'єднання з внутрішніми. Таке з'єднання називається з'єднанням делегації.

Нижче наведемо ту частину коду, яка виконує вищеперераховані дії.

```
namespace Disk
{
```

```
    /// <summary>
    /// Клас кодера циклічного коду
    /// </summary>
    public class CicleCodeEncoder : CicleCodeBase
    {
        #region Construction & Destruction
        /// <summary>
        /// Конструктор кодера за замовчуванням
        /// </summary>
        public CicleCodeEncoder()
        {
// Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }
/// <summary>
/// Базовий конструктор кодера
/// </summary>
/// <param name="dataCount">Кількість основних томів</param>
/// <param name="eccCount">Кількість томів для відновлення</param>
        public CicleCodeEncoder(int dataCount, int eccCount)
        {
// Установка конфігурації кодера
            SetConfig(dataCount, eccCount, (int)CicleCodeType.Alternative);
// Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }
/// <summary>
/// Розширений конструктор кодера
/// </summary>
/// <param name="dataCount">Кількість основних томів</param>
/// <param name="eccCount">Кількість томів для відновлення</param>
/// <param name="codecType">Тип кодера циклічного коду (по типу матриці)</param>
        public CicleCodeEncoder(int dataCount, int eccCount, int codecType)
        {
// Установка конфігурації кодера
            SetConfig(dataCount, eccCount, codecType);
// Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }
        #endregion Construction & Destruction
        #region Public Operations
/// <summary>
```

```
/// Установка конфігурації кодера
/// </summary>
/// <param name="dataCount">Кількість основних томів</param>
/// <param name="eccCount">Кількість томів для відновлення</param>
/// <param name="codecType">Тип кодека кодера циклічного коду
/// (по типу матриці)</param>
/// <returns>Булевський прапор операції установки конфігурації</returns>
        public bool SetConfig(int dataCount, int eccCount, int codecType)
        {
            int maxVolCount;
// Установлюємо константи, що відповідають обраному режиму
            if (codecType == (int)CicleCodeType.Dispersal)
            {
                maxVolCount = (int)CicleCodeConst.MaxVolCountDisp;
            } else
            {
                maxVolCount = (int)CicleCodeConst.MaxVolCountAlt;
            }
// Перевіряємо конфігурацію на коректність
            if (
                    (dataCount > 0)
                 &&
                    (eccCount > 0)
                 &&
                    ((dataCount + eccCount) <= maxVolCount)
                )
            {
// Якщо основна конфігурація змінилася – сповіщаємо про це
                if (
                        (dataCount != this.n)
                     ||
                        (eccCount != this.m)
                     ||
                        (codecType != this.eCicleCodeType)
                    )
                {
                    this.mainConfigChanged = true;
                }
// Зберігаємо конфігурацію
                this.n = dataCount;
                this.m = eccCount;
                this.eCicleCodeType = codecType;
```

```
// Також перераховуємо кількість ітерацій всіх стадій підготовки
                double n = this.n;
                double m = this.m;
// Нормалізуємо значення для розрахунку, щоб уникнути переповнення змінних
                NormalizeNM(ref n, ref m);
// Кількість ітерацій на першій стадії залежить від типу використовуваної матриці
                if (this.eCicleCodeType == (int)CicleCodeType.Alternative)
                {
                    this.iterOfFirstStage = m;
                } else
                {
           this.iterOfFirstStage =((n * m)* n)+(n * ((n + m) + (n * (n + m))));
                }
                this.iterOfSecondStage = 0;
// У кодери немає інвертування матриці
                this.configIsOK = true;
            } else
            {
//...указуємо на помилку конфігурації
                this.configIsOK = false;
            }
            return this.configIsOK;
        }
/// <summary>
/// Метод множення матриці кодування на вхідний прологарифмований вектор
/// </summary>
/// <param name="dataLog">Прологарифмований вхідний вектор (вихідні дані)</param>
/// <param name="ecc">Вихідний вектор (надлишкові дані)</param>
/// <returns>Булевський прапор результату операції</returns>
        public bool Process(int[] dataLog, ref int[] ecc)
        {
// Якщо кодер зконфігуровано некоректно, обробка неможлива!
            if (!this.configIsOK)
            {
                return false;
            }
// Копіюємо покажчик на масив експонент для скорочення часу обігу
            int[] GF16Exp = this.eGF16.GFExpTable;
// Обчислення результату множення матриці на вектор
            for (int i = 0; i < this.m; i++)
            {
                int mulSum = 0;
```

```
// Сума добутку рядка матриці на стовпець
                int i_n = i * this.n;
// Зсув у масиві до елементів i-ой рядка
                for (int j = 0; j < this.n; j++)
                {
                    mulSum ^= GF16Exp[this.FLog[i_n + j] + dataLog[j]];
                }
                ecc[i] = mulSum;
            }
            return true;
        }
        #endregion Public Operations
        #region Private Operations
/// <summary>
/// Заповнення матриці Вандермонда даними
/// </summary>
        protected override void FillFLog()
        {
// Якщо основна конфігурація змінилася...
            if (this.mainConfigChanged)
            {
                if (this.eCicleCodeType == (int)CicleCodeType.Dispersal)
                {
//...робимо формування дисперсної матриці "D"
                    if (!MakeDispersalMatrix())
                    {
// Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;
// Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;
// Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();
                        return;
                    }
                } else
                {
//...робимо формування альтернативного заповнення матриці "A"
                    if (!MakeAlternativeMatrix())
                    {
// Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;
// Активуємо індикатор актуального стану змінних-членів
```

```
                        this.finished = true;
// Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();
                        return;
                    }
                }
// Виділяємо пам'ять під матрицю "FLog"
                this.FLog = new int[this.m * this.n];
// Заповнюємо матрицю кодування
                for (int i = 0; i < this.m; i++)
                {
// Зсув у масиві до елементів i-ой рядка
                    int i_n = i * this.n;
// Залежно від типу кодера беремо дані з відповідного масиву
                    if (this.eCicleCodeType == (int)CicleCodeType.Dispersal)
                    {
                        // Формування рядка в матриці кодування
                        for (int j = 0; j < this.n; j++)
                        {
// У матрицю кодування поміщаємо логарифми її вихідних елементів
// (для прискорення множення матриці на вектор)
this.FLog[i_n + j] = this.eGF16.Log(this.D[((this.n + i) * this.n) + j]);
                        }
                    } else
                    {
// Формування рядка в матриці кодування
                        for (int j = 0; j < this.n; j++)
                        {
                            int idx = i_n + j;
// У матрицю кодування поміщаємо логарифми її вихідних елементів
// (для прискорення множення матриці на вектор)
                            this.FLog[idx] = this.eGF16.Log(this.A[idx]);
                        }
                    }
// У випадку, якщо потрібна постановка на паузу, подію "executeEvent"
// буде скинуто, і будемо на паузі аж до його появи
                    ManualResetEvent.WaitAll(this.executeEvent);
// Якщо зазначено, що потрібно вийти з потоку – виходимо
                    if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                    {
// Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;
```

```
// Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;
// Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();
                        return;
                    }
                }

// Якщо є передплата на делегата завершення...
                if (OnCicleCodeMatrixFormingFinish != null)
                {
//...повідомляємо, що екземпляр класу готовий до роботи
                    OnCicleCodeMatrixFormingFinish();
                }

//...і скидаємо прапор
                this.mainConfigChanged = false;
            }

// Якщо є передплата на делегата завершення...
            if (OnCicleCodeMatrixFormingFinish != null)
            {
//...повідомляємо, що екземпляр класу готовий до роботи
                OnCicleCodeMatrixFormingFinish();
            }
            // Активуємо індикатор актуального стану змінних-членів
            this.finished = true;
            // Установлюємо подію завершення обробки
            this.finishedEvent[0].Set();
        }
        #endregion Private Operations
    }
}
```

### 4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм Madryga. Алгоритм Madryga складається із двох вкладених циклів. Зовнішній цикл повторюється вісім разів (для гарантії

надійності число циклів можна збільшити) і полягає в застосуванні внутрішнього циклу до відкритого тексту. Внутрішній цикл перетворює відкритий текст у шифртекст і виконується однократно над кожним 8-бітовим блоком (байтом) відкритого тексту. Таким чином, весь відкритий текст послідовно вісім разів обробляється алгоритмом. Ітерація внутрішнього циклу оперує з 3-байтовим вікном даних, називаним робочим кадром (рисунок 4.3). Це вікно зрушується на 1 байт за ітерацію. (При роботі з останніми 2 байтами дані покладаються циклічно замкнутими). Перші два байти робочого кадру циклічно зрушуються на змінне число позицій, а для останнього байта виконується операція XOR з декількома бітами ключа. У міру переміщення робочого кадру всі байти послідовно циклічно зрушуються й піддаються операції XOR із частинами ключа. Послідовні циклічні зрушення перемішують результати попередніх операцій XOR і циклічного зрушення, причому на циклічне зрушення впливають результати XOR. Завдяки цьому процес у цілому оборотний.

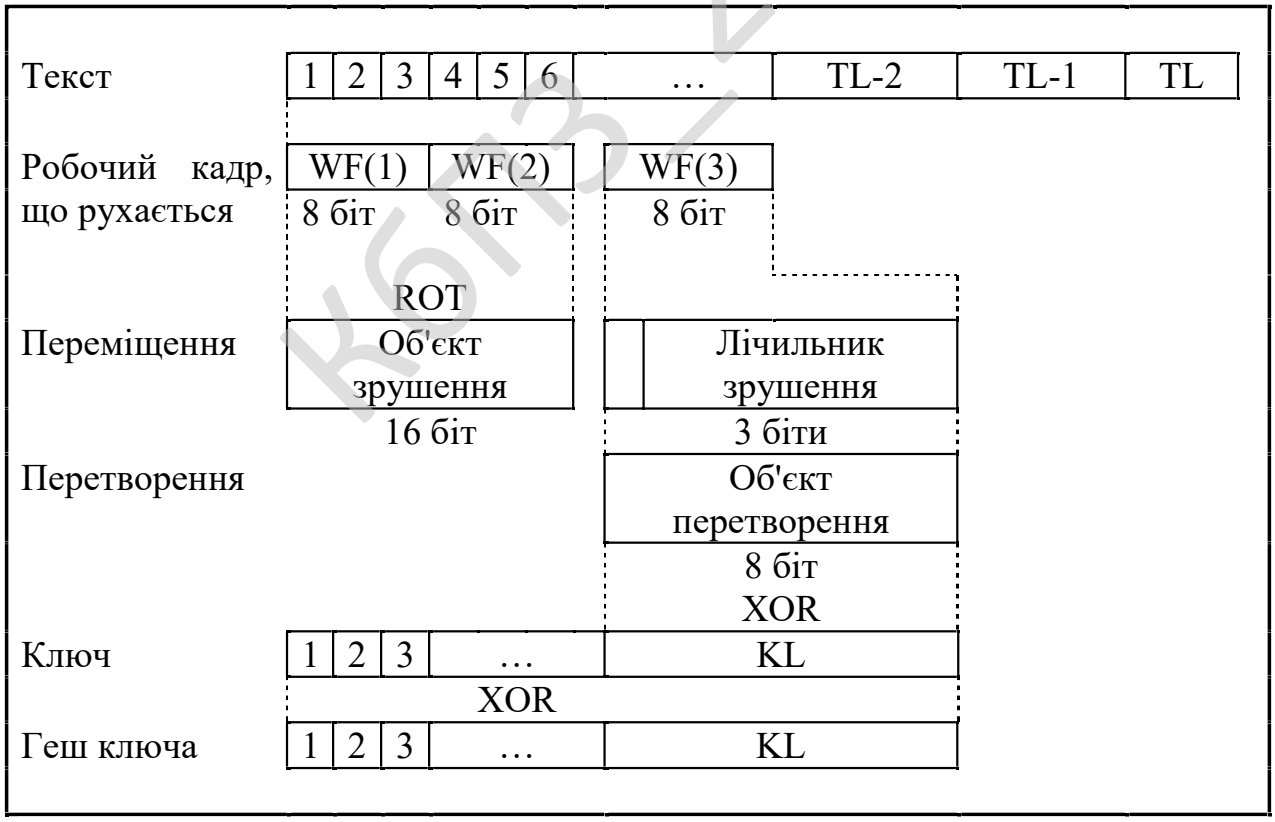


Рисунок 4.3 – Одна ітерація алгоритму Madryga

Оскільки кожний байт даних впливає на два байти ліворуч і на один байт праворуч від себе, після восьми проходів кожний байт шифртексту залежить від 16 байтів ліворуч і 8 байтів праворуч.

При шифруванні кожна ітерація внутрішнього циклу встановлює робочий кадр на передостанній байт відкритого тексту й циклічно переміщає його до третього з кінця байту відкритого тексту. Спочатку весь ключ піддається операції XOR з випадковою константою й потім циклічно зрушується вправо на 3 біти (ключ і дані рухаються в різних напрямках, щоб мінімізувати надлишкові операції з бітами ключа). Молодші три біти молодшого байта робочого кадру зберігаються, вони визначають циклічне зрушення інших двох байтів. Далі конкатенація двох старших байтом циклічно зрушується вліво на змінне число біт (від 0 до 7). Потім над молодшим байтом робочого кадру виконується операція XOR з молодшим байтом ключа. Нарешті робочий кадр зміщається вправо на один байт і весь процес повторюється.

Випадкова константа призначена для перетворення ключа в псевдовипадкову послідовність. Довжина константи повинна бути рівній довжині ключа. При обміні даними абоненти повинні користуватися однієї й тією же константою. Для 64-бітового ключа Madryga рекомендує константу 0x0fle2d3c4b5a6978.

При розшифруванні процес повторюється у зворотному порядку. У кожній ітерації внутрішнього циклу робочий кадр установлюється на байт, третій ліворуч від останнього байта шифртексту, і циклічно зрушується у зворотному напрямку до байта, розташованого на 2 байти уліво відносно останнього байта шифртексту. 2 байти шифртексту в процесі циклічно зрушуються вправо, а ключ – уліво. Після циклічних зрушень виконується операція XOR.

# 5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено інтерфейс програмного забезпечення, розробленого у результаті виконання бакалаврської дипломної роботи. Розроблене програмне забезпечення системи підвищення надійності функціонування ЦОД складається з наступних функціональних блоків:

– Навігаційне меню: Додатки; ЦОД; Функції; Довідка.

– Вікно виведення результату роботи системи.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Функціональних кнопок ПЗ: Збереження даних; Відновлення даних; Скасування; Пошук; Виправлення; Налаштування.

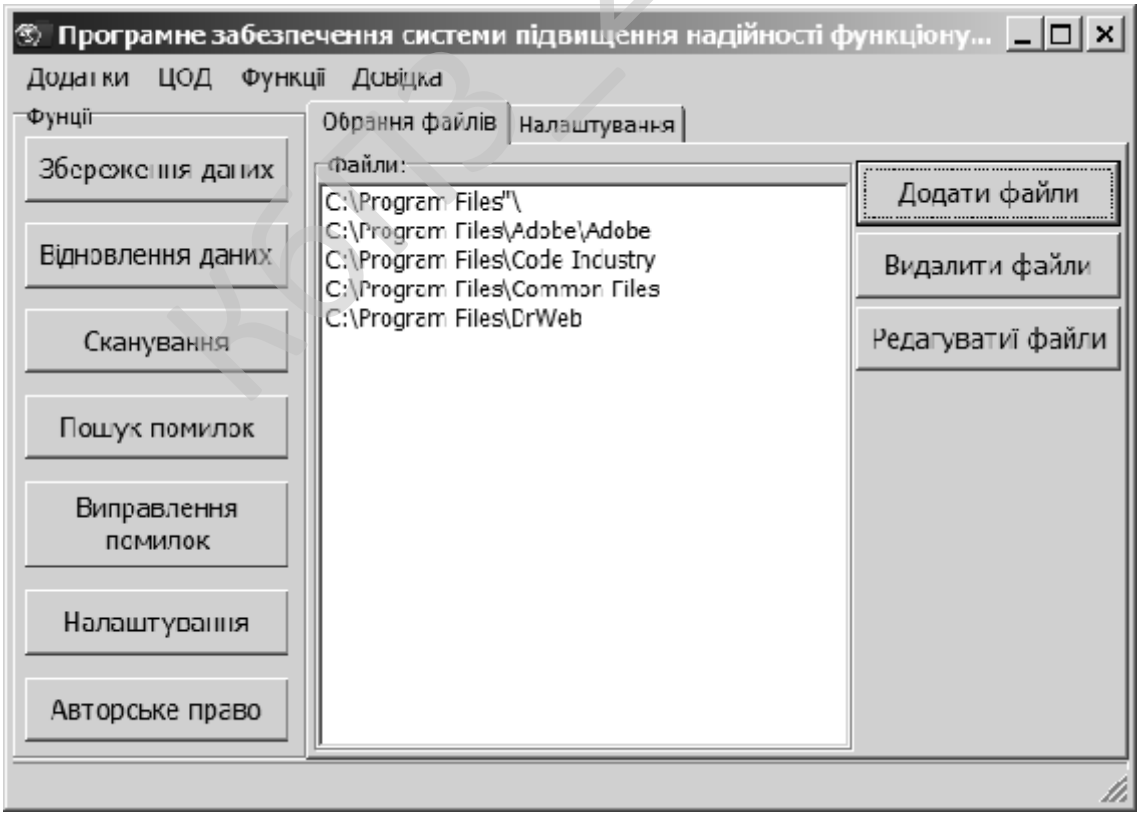


Рисунок 5.1 – Головне вікно розробленого ПЗ

Для перегляду короткої довідки про програму слід натиснути на основному вікні кнопку авторського права, після чого на екрані з’явиться вікно показане на рисунку 5.2.

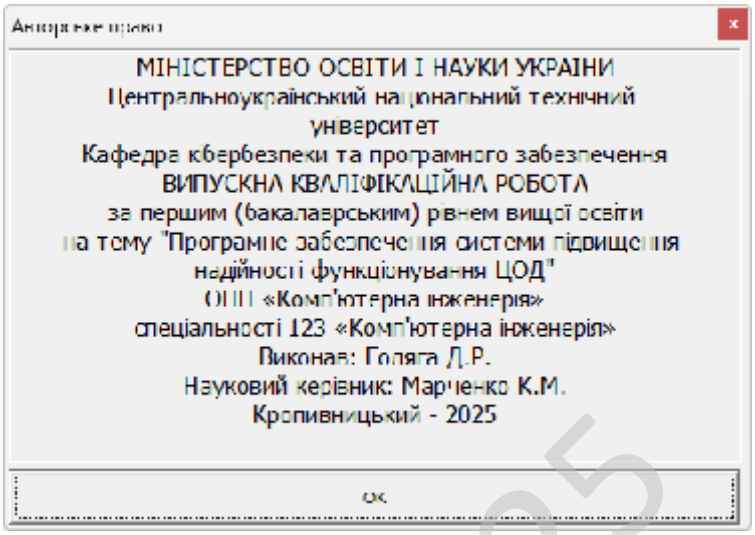


Рисунок 5.2 – Вікно розробника ПЗ

Під час роботи над програмою було проведено тестування програмного забезпечення, тобто технічне дослідження, призначене для виявлення інформації про якість продукту відносно контексту, в якому воно має використовуватись.

Тестування включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою їх оцінки.

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

Оскільки число можливих тестів для програмних компонент практично нескінченне, тому стратегія тестування полягала в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів.

Як результат ПЗ тестувалось стандартним виконанням програми з метою виявлення помилок або інших дефектів.

Проводилось тестування форматом чорної скриньки. Основне місце програми тестів «чорної скриньки» – інтерфейс ПЗ. Відомі: функції програми. Досліджується: робота кожної функції на всій області визначення.

Ці тести демонструють:

– Як виконуються функції програми.

– Як приймаються вихідні дані.

– Як виробляються результати.

– Як зберігається цілісність зовнішньої інформації.

При тестуванні «чорної скриньки» розглядаються системні характеристики програм, ігнорується їхня внутрішня логічна структура. Вичерпне тестування, як правило, неможливе.

Наприклад, якщо в програмі 10 вхідних величин і кожна приймає по 10 значень, то кількість тестових варіантів становитиме 10^10. Тестування «чорної скриньки» не реагує на багато особливостей програмних помилок.

Тестування «чорної скриньки» (функціональне тестування) дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми.

Програмний виріб тут розглядається як «чорна скринька», чию поведінку можна визначити тільки дослідженням його входів та відповідних виходів. При такому підході бажано мати:

– Набір, утворений такими вхідними даними, які призводять до аномалій у поведінці програми (назвемо його ITc).

– Набір, утворений такими вхідними даними, які демонструють дефекти програми (назвемо його OT).

Будь-який спосіб тестування «чорної скриньки» повинен:

– Виявити такі вхідні дані, які з високою ймовірністю належать набору ITc;

– Сформулювати такі очікувані результати, які з високою імовірністю є елементами набору OT.

Принцип «чорної скриньки» не альтернативний принципу «білої скриньки». Скоріше це доповнює підхід, який виявляє інший клас помилок.

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

– Некоректних чи відсутніх функцій;

– Помилок інтерфейсу;

– Помилок у зовнішніх структурах даних або в доступі до зовнішньої бази даних;

– Помилок характеристик (необхідна ємність пам'яті і т.д.);

– Помилок ініціалізації та завершення.

Обрано умови розповсюдження – Shareware.

Під умовно-безплатним програмним забезпеченням можна розуміти спосіб або метод розповсюдження комерційного ПЗ на ринку (тобто на шляху до кінцевого користувача), при якому випробувачеві пропонується обмежена за можливостями (не повнофункціональна або демонстраційна версія), терміном дії (тріал версія) або версія з вбудованим набридливим нагадуванням про необхідність оплати використання програми.

В угоді про використання (ліцензії для кінцевого користувача, EULA) також може бути обумовлена заборона на комерційне або професійне (не тестове) її використання.

Основний принцип умовно-безплатного ПЗ – «спробуй, перш ніж купити» (try before you buy). ПЗ що поширюється як умовно-безплатний, надається користувачам безоплатно.

Звичайно користувач платить тільки за час завантаження файлів через Інтернет або за носій (CD диск, флешку, ключ).

Протягом певного терміну, що становить зазвичай тридцять днів, він може користуватися програмою, тестувати її, освоювати її можливості.

Якщо після закінчення цього терміну користувач вирішить продовжити використання ПЗ, він зобов'язаний купити його (зареєструватися), заплативши авторові певну суму.

В іншому випадку користувач повинен припинити використання ПЗ та видалити його зі свого комп'ютера.

# 6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи підвищення надійності функціонування ЦОД.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем підвищення надійності функціонування ЦОД.

– Досліджена система підвищення надійності функціонування ЦОД.

– На основі отриманих результатів досліджень створена програмна реалізація системи підвищення надійності функціонування ЦОД.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання підвищення надійності функціонування ЦОД.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Visual C#. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи підвищення надійності функціонування ЦОД. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне

програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм Madryga.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Mark S. Merkow. Practical Security for Agile and DevOps. CRC Press. 2022. 236 p

2. Derek Fisher. Application Security Program Handbook. Manning Publications. 2021. 155 p.

3. Cameron Wyatt PH.D. Kali Linux Tutorial. Independently published. 2021. 60 p.

4. Alex Matrosov, Eugene Rodionov, Sergey Bratus. Rootkits and Bootkits. No Starch Press. 2019. 450 p.

5. Lakhno, V., Malyukov, V., Smirnov, O., Bebeshko, B., Chubaievskiy, V., Zhumadilova, M., Malyukova, I., Smirnov, S. «Multifactorial Model for Targeted Attacks Counteracting Within the Framework of a Multi-Step Quality Game with Fuzzy Information». *8th International Symposium on Intelligent Informatics, ISI 2023*, 2025. vol 389. pp 377-389. Springer, Singapore.

6. Kuznetsov, O., Frontoni, E., Kryvinska, N., Chevardin, V., Smirnov, O. «Wireless Network Encryption Stream Ciphers, Computational Modeling, and Security Analysis». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 379–402.

7. Kuznetsov, O., Frontoni, E., Kryvinska, N., Smirnov, O., Imoize, G.L. «Computational Modeling of Enhanced Spread Spectrum Codes for Asynchronous Wireless Communication». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 403–447.

8. Kuznetsov, O., Kryvinska, N., Ilchenko, O., Smirnova, T., Ulianovska, Y. «Comparative Analysis of Cryptocurrency Trading Platforms Using the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, 2023, 3628, pp. 106-115.

9. Akhalaia, G., Iavich, M., Iashvili, G., Prysiazhnyy, D., Smirnova, T. «Secure Encrypted Connection on Georgian Website». *CEUR Workshop Proceedings,* 2023, 3550, pp. 313-320.

10. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». *Advanced Information Systems*, 2023, 7(2), pp. 49-56

11. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». *CEUR Workshop Proceedings,* Volume 3530, 2023, pp. 256-265.

12. Kuznetsov, O., Kandiy, S., Frontoni, E., Smirnov, O. «Trade-offs in Post-Quantum Cryptography: A Comparative Assessment of BIKE, HQC, and Classic McEliece». *CEUR Workshop Proceedings,* Volume 3504, 2023, pp. 1-11.

13. Smirnov, O., Neskorodieva, T., Fedorov, E., Rudakov, K., Neskorodieva, A. «Method Detection Audit Data Anomalies on Basis Restricted Cauchy Machine» *CEUR Workshop Proceedings,* Volume 3187, 2022,

14. Smirnov, O., Lakhno, V., Akhmetov, B., Chubaievskyi, V., Khorolska, K., Bebeshko, B. «Selection of a Rational Composition of İnformation Protection Means Using a Genetic Algorithm». *In: Rajakumar, G., Du, KL., Vuppalapati, C., Beligiannis, G.N. (eds) Intelligent Communication Technologies and Virtual Mobile Networks. Lecture Notes on Data Engineering and Communications Technologies*, vol 131. 2023. Springer, Singapore. pp. 21-34.

15. Smirnov O.A., Al-Oraiqat A.M., Ulichev O.S., Meleshko Ye.V., Al-Rawashdeh H.S., Polishchuk L.I. «Modeling strategies for information influence dissemination in social networks». *Journal of Ambient Intelligence and Humanized Computing* Volume 13, Issue 5. Springer, Cham. 2022, pp. 2463-2477.

16. Smirnov O., Kuznetsov A., Zhora V., Onikiychuk A., Pieshkova O. «Hiding Messages in Audio Files Using Direct Spread Spectrum». *11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing*

*Systems: Technology and Applications, IDAACS 2021,* Cracow, Poland, 22-25 September 2021. P. 414-418

17. Smirnov O., Kuznetsov A., Lokotkova I., Kuznetsova T., Florov S., Lebid O. «Using Orthogonal Signals to Hide Information in Images». *4 IEEE International Conference on Advanced Information and Communication Technologies (AICT) - 2021*, Lviv, Ukraine, September 21-25, 2021. P. 255-260.

18. Smirnov O., Kuznetsov A., Girzheva O., Kiian A., Nakisko O., Kuznetsova T. «Advanced Code-Based Electronic Digital Signature Scheme». *2020 IEEE International Conference on Problems of Infocommunications Science and Technology, PIC S and T 2020,* Kharkiv, 6 October 2020-9 October 2020, P. 358-362.

19. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova K. «Data hiding scheme based on spread sequence addressing». *CEUR Workshop Proceedings* Volume 2805, 2020, Pages *44-58.*

20. Smirnov, O., Kuznetsov, A., Potii, O., Poluyanenko, N., Stelnyk, I., Mialkovsky, D. «Combining and filtering functions in the framework of nonlinear-feedback shift register». *International Journal of Computing*; 2020, Volume 19, Issue 2 – Research Institute for Intelligent Computer Systems – 2020. – P. 247-256.

21. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». *CEUR Workshop Proceedings*. Volume 2740, 2020, Pages 102-114.

22. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and cybersecurity of virtual cloud resources». *Journal of theoretical and applied information technology* Vol.98. No 21, 2020, P. 3334-3346.

23. Smirnov O., Kuznetsov A., Arischenko A., Chepurko I., Onikiychuk A., Kuznetsova T. «Pseudorandom sequences for spread spectrum image steganography». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 122-131.

24. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New technique for data hiding in cover images using adaptively generated pseudorandom sequences». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 1-14.

25. Smirnov O., Lutsenko M., Kuznetsov A., Kiian A., Kuznetsova T., «Biometric cryptosystems: overview, state-of-the-art and perspective directions». *Lecture Notes in Networks and Systems*, vol 152. Springer, Cham. 2021, pp 66-84.

26. Smirnov O., Kuznetsov A., Onikiychuk A., Makushenko T., Anisimova O., Arischenko A. «Adaptive pseudo-random sequence generation for spread spectrum image steganography». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, Ukraine, Kyiv, May 14-18. 2020. P. 161-165.

27. Smirnov O., Kuznetsov A., Kiian A., Babenko V., Perevozova I.,Chepurko I. «New Approach to the Implementation of Post-Quantum Digital Signature Scheme». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 166-171.

28. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

29. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

30. Smirnov, O., Markovets, O. Vovk, N., Turchyn, Y., «Model of informational support for social network administrators' content creation». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 125-136.

31. Smirnov, O., Shekhanin, K., Kuznetsov, A., Krasnobayev, V. «Detecting Hidden Information in FAT». *International Journal of Computer Network and Information Security (IJCNIS)*. Vol. 12, No. 3, 2020. PP.33-43.

32. Smirnov, O., Kuznetsov, A., Gorbacheva, L., Babenko, V., «Hiding data in images using a pseudo-random sequence», *CEUR Workshop Proceedings* Volume 2608, 2020, Pages 646-660.

33. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

34. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.

35. Smirnov, O., Ulichev, O., Meleshko, Y., Khokh, V., Goncharenko, I. «Method of Choosing Objects for Informational Influence in Social Networks during Information Campaign Based on the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, Vol 2588, P. 215-227, 2019.

36. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». *CEUR Workshop Proceedings*, Vol 2588, P. 90-106, 2019.

37. Smirnov, O., Kuznetsov, A., Kiian, A., Gorbenko, Y., Cherep, O., Bexhter L. «Code-based Pseudorandom Generator for the Post-Quantum Period», *2019 IEEE International Conference on Advanced Trends in Information Theory (IEEE ATIT 2019).* 18.12.19-20.12.19 Kyiv Ukraine. P. 204 – 209.

38. Smirnov, O., Kuznetsov, A., Nariezhnii, O., Stelnyk, S., Kokhanovska, T., Kuznetsova T., «Side Channel Attack on a Quantum Random Number Generator», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced*

*Computing Systems: Technology and Applications, IDAACS 2019*; Metz; France; 18 - 21 September 2019. P.713-718.

39. Kuznetsova, T., «Code-Based Schemes for Post-Quantum Digital Signatures», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019;* Metz; France; 18-21 September 2019. P. 707-712.

40. Smirnov, O., Kuznetsov, A., Stefanovych, O., Gorbenko, Y., Krasnobaev, V., Kuznetsova K. «Information Hiding Using 3D-Printing Technology», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019*; Metz; France; 18-21 September 2019. P.701-706.

41. Smirnov, O., Hu, Z., Vasiliu, Y., Sydorenko, V., Polishchuk, Y., «Abstract Model of Eavesdropper and Overview on Attacks in Quantum Cryptography Systems», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019*; Metz; France; 18-21 September 2019. P.399-405.

42. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», *2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019*/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

43. Smirnov, O., Kuznetsov, A., Kiian, A., Babenko, B., Zhosan, H., Prokopovych-Tkachenko, D., «Soft Decoding Method for Turbo-Productive Codes», *2019 3rd International Conference on Advanced Information and Communications Technologies, AICT 2019*, Lviv, Ukraine, 2-6 July, 2019, P. 129-134.

44. Smirnov, O., Kuznetsov, A., Kiian, A., Zamula, A., Rudenko, S., Hryhorenko, V., «Variance Analysis of Networks Traffic for Intrusion Detection in Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 353-358.

45. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 347-352.

46. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 618-629.

47. Smirnov, O., Kuznetsov, A., Kiian, A., Kuznetsova, K., Ivko, T., Prokopovych-Tkachenko, D., «Soft Decoding Based on Ordered Subsets of Verification Equations of Turbo-Productive Codes», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 873-884.

48. Smirnov A.A., Kuznetsov A.A., Danilenko D.A., Berezovsky A., «The statistical analysis of a network traffic for the intrusion detection and prevention systems», *Telecommunications and Radio Engineering*. – Volume 74, Issue 1. – Begel House Inc. – 2015. – P. 61-78.

49. Ткаченко, О., Ільєнко, А., Улічев, О., Мелешко, Є., Смірнов, О. «Правові засади поширення інформаційних впливів в соціальних мережах». *Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка»,* 2024. № 2(26), С. 170–188.

50. Смірнова Т.В., Конопліцька-Слободенюк О.К., Буравченко К.О., Смірнов С.А., Кравчук О.В., Козірова Н.Л., Смірнов О.А. «Дослідження технологій забезпечення кібербезпеки хмарних сервісів IaaS, PaaS та SaaS». *Кібербезпека: освіта, наука, техніка.* 2024. №4(24), С. 6-27.

51. Вінтенко, Б., Миронець, І., Смірнов, О., Кравчук, О., Козірова, Н., Савеленко, Г., Коваленко, А. «Дослідження вимог та аналіз кібербезпеки програмного забезпечення інформаційно-керуючих систем АЕС, важливих для безпеки». *Кібербезпека: освіта, наука, техніка.* 2024. №3(23), С. 111-131.

Додаток А
(обов’язковий)

**Технічне завдання**

**Зміст**

## 1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи підвищення надійності функціонування ЦОД.

## 2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 47-02 від 17.01.2025 року).

## 3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи підвищення надійності функціонування ЦОД.

## 4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

## 5 Технічні вимоги

### 5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

### 5.2 Показники призначення

Система повинна забезпечувати:

– системи підвищення надійності функціонування ЦОД;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

### 5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

### 5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

### 5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

### 5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

### 5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпечення.

### 5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.



#### 5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

#### 5.8.2 Мова програмування

Середовище Visual C#.

### 5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

### 5.8.4 Вихідні дані

Робоча програма.

## 6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

## 7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 79 аркушів.

## 8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

## 9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2025 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 3.06.2025 р.

Додаток Б

(обов’язковий)

**Міністерство освіти і науки України**

**Центральноукраїнський національний технічний університет**

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

_____ Марченко К.М.

***Програмне забезпечення системи підвищення надійності функціонування ЦОД***



Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 50

Літера: РП

Кропивницький – 2025 року

**Файл MainForm.cs - головне вікно програми**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;

namespace DataCenterReliability
{
    class Program
    {
        static void Main(string[] args)
        {
            DataCenter dc = new DataCenter();
            dc.Initialize();
            dc.MonitorSystem();
        }
    }

    class DataCenter
    {
        private List<Server> servers;
        private PowerSupply powerSupply;
        private CoolingSystem coolingSystem;
        private NetworkSystem networkSystem;
        private SecuritySystem securitySystem;

        public void Initialize()
        {
            servers = new List<Server>();
            for (int i = 0; i < 10; i++)
            {
                servers.Add(new Server(i));
            }
            powerSupply = new PowerSupply();
            coolingSystem = new CoolingSystem();
            networkSystem = new NetworkSystem();
            securitySystem = new SecuritySystem();
        }

        public void MonitorSystem()
        {
            while (true)
            {
                foreach (var server in servers)
                {
                    server.CheckHealth();
                }
                powerSupply.CheckStatus();
                coolingSystem.CheckTemperature();
                networkSystem.CheckConnectivity();
                securitySystem.CheckSecurity();
                Thread.Sleep(5000);
            }
        }
    }

    class Server
    {
        private int id;
        private bool isActive;
        private Random random;

        public Server(int id)
        {
            this.id = id;
            this.isActive = true;
            this.random = new Random();
```

```
    }

    public void CheckHealth()
    {
        isActive = random.Next(0, 10) > 1;
        if (!isActive)
        {
            RestartServer();
        }
    }

    private void RestartServer()
    {
        isActive = true;
    }
}

class PowerSupply
{
    private bool isOperational;
    private Random random;

    public PowerSupply()
    {
        this.isOperational = true;
        this.random = new Random();
    }

    public void CheckStatus()
    {
        isOperational = random.Next(0, 20) > 2;
        if (!isOperational)
        {
            RestorePower();
        }
    }

    private void RestorePower()
    {
        isOperational = true;
    }
}

class CoolingSystem
{
    private int temperature;
    private Random random;

    public CoolingSystem()
    {
        this.temperature = 22;
        this.random = new Random();
    }

    public void CheckTemperature()
    {
        temperature = random.Next(18, 28);
        if (temperature > 25)
        {
            ActivateCooling();
        }
    }

    private void ActivateCooling()
    {
        temperature = 22;
    }
}
```

```
    class NetworkSystem
    {
        private bool isConnected;
        private Random random;

        public NetworkSystem()
        {
            this.isConnected = true;
            this.random = new Random();
        }

        public void CheckConnectivity()
        {
            isConnected = random.Next(0, 15) > 1;
            if (!isConnected)
            {
                Reconnect();
            }
        }

        private void Reconnect()
        {
            isConnected = true;
        }
    }

    class SecuritySystem
    {
        private bool breachDetected;
        private Random random;

        public SecuritySystem()
        {
            this.breachDetected = false;
            this.random = new Random();
        }

        public void CheckSecurity()
        {
            breachDetected = random.Next(0, 50) == 1;
            if (breachDetected)
            {
                TriggerAlarm();
            }
        }

        private void TriggerAlarm()
        {
            breachDetected = false;
        }
    }
}
```

**Файл CicleCodeDecoder.cs - декодер циклічного коду для підвищення надійності функціонування ЦОД**

```
using System;
using System.Threading;

namespace RecoveryDisk
{
    /// <summary>
    /// Клас декодера циклічного коду для підвищення надійності функціонування
ЦОД
    /// </summary>
    public class CicleCodeDecoder : CicleCodeBase
    {
        #region Data

        // Масив булевських ознак "рядок матриці "FLog" тривіальна?"
        private bool[] FLogRowIsTrivial;

        // Список порядкових номерів наявних томів (нумерація з нуля)
        private int[] volList;

        #endregion Data

        #region Construction & Destruction

        /// <summary>
        /// Конструктор декодера за замовчуванням
        /// </summary>
        public CicleCodeDecoder()
        {
            // Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }

        /// <summary>
        /// Базовий конструктор декодера
        /// </summary>
        /// <param name="dataCount">Кількість основних томів</param>
        /// <param name="eccCount">Кількість томів для відновлення</param>
        /// <param name="volList">Список порядкових номерів наявних
томів</param>
        public CicleCodeDecoder(int dataCount, int eccCount, int[] volList)
        {
            // Установка конфігурації кодера
            SetConfig(dataCount, eccCount, volList,
(int)CicleCodeType.Alternative);

            // Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }

        /// <summary>
        /// Розширений конструктор декодера
        /// </summary>
        /// <param name="dataCount">Кількість основних томів</param>
        /// <param name="eccCount">Кількість томів для відновлення</param>
        /// <param name="volList">Список порядкових номерів наявних
томів</param>
        /// <param name="codecType">Тип кодера циклічного коду для підвищення
надійності функціонування ЦОД (по типу матриці)</param>
        public CicleCodeDecoder(int dataCount, int eccCount, int[] volList, int
codecType)
        {
            // Установка конфігурації кодера
            SetConfig(dataCount, eccCount, volList, codecType);
```

```
            // Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }

        #endregion Construction & Destruction

        #region Public Operations

        /// <summary>
        /// Установка конфігурації декодера
        /// </summary>
        /// <param name="dataCount">Кількість основних томів</param>
        /// <param name="eccCount">Кількість томів для відновлення</param>
        /// <param name="volList">Список порядкових номерів наявних
томів</param>
        /// <param name="codecType">Тип кодека кодера циклічного коду для
підвищення надійності функціонування ЦОД (по типу матриці)</param>
        /// <returns>Булевський прапор операції установки конфігурації</returns>
        public bool SetConfig(int dataCount, int eccCount, int[] volList, int
codecType)
        {
            int maxVolCount;

            // Установлюємо константи, що відповідають обраному режиму
            if (codecType == (int)CicleCodeType.Dispersal)
            {
                maxVolCount = (int)CicleCodeConst.MaxVolCountDisp;

            } else
            {
                maxVolCount = (int)CicleCodeConst.MaxVolCountAlt;
            }

            // Перевіряємо конфігурацію на коректність
            if (
                    (dataCount > 0)
                 &&
                    (eccCount > 0)
                 &&
                    ((dataCount + eccCount) <= maxVolCount)
                 &&
                    (volList.Length >= dataCount)
                )
            {
                // Якщо основна конфігурація змінилася - сповіщаємо про це
                if (
                        (dataCount != this.n)
                     ||
                        (eccCount != this.m)
                     ||
                        (codecType != this.eCicleCodeType)
                    )
                {
                    this.mainConfigChanged = true;
                }

                // Зберігаємо конфігурацію
                this.n = dataCount;
                this.m = eccCount;
                this.eCicleCodeType = codecType;

                // Також перераховуємо кількість ітерацій всіх стадій підготовки
                double n = this.n;
                double m = this.m;

                // Нормалізуємо значення для розрахунку, щоб уникнути
переповнення змінних
                NormalizeNM(ref n, ref m);
```

```
                // Кількість ітерацій, що відслідковуються прогресом, на першій
стадії
                // залежить від типу використовуваної матриці
                if (this.eCicleCodeType == (int)CicleCodeType.Alternative)
                {
                    this.iterOfFirstStage = m;

                } else
                {
                    this.iterOfFirstStage = ((n * m) * n) + (n * ((n + m) + (n *
(n + m))));
                }

                this.iterOfSecondStage = (n * (((n - 1) * (n - 1)) + (n * n)));

                // Виділяємо пам'ять під масив булевських ознак "рядок матриці
"FLog" тривіальна?"
                this.FLogRowIsTrivial = new bool[dataCount];

                // Зберігаємо список наявних томів
                this.volList = volList;

                this.configIsOK = true;

            } else
            {
                //...указуємо на помилку конфігурації
                this.configIsOK = false;
            }

            return this.configIsOK;
        }

        /// <summary>
        /// Метод множення матриці кодування на вхідний прологарифмований вектор
        /// </summary>
        /// <param name="dataEccLog">Прологарифмований вхідний вектор (дані +
ecc)</param>
        /// <param name="data">Вихідний вектор (відновлені вихідні дані)</param>
        /// <returns>Булевський прапор результату операції</returns>
        public bool Process(int[] dataEccLog, ref int[] data)
        {
            // Якщо кодер зконфігуровано некоректно, обробка неможлива!
            if (!this.configIsOK)
            {
                return false;
            }

            // Копіюємо покажчик на масив експонент для скорочення часу обігу
            int[] GF16Exp = this.eGF16.GFExpTable;

            // Обчислення результату множення матриці на вектор
            for (int i = 0; i < this.n; i++)
            {
                // Якщо поточний рядок матриці не є тривіальної, робимо обробку
                if (!this.FLogRowIsTrivial[i])
                {
                    int mulSum = 0;     // Сума добутку рядка матриці на
стовпець
                    int i_n = i * this.n; // Зсув у масиві до елементів i-ой
рядка

                    for (int j = 0; j < this.n; j++)
                    {
                        mulSum ^= GF16Exp[this.FLog[i_n + j] + dataEccLog[j]];
                    }

                    data[i] = mulSum;
```

```
                } else
                {
                    data[i] = GF16Exp[dataEccLog[i]];
                }
            }

            return true;
        }

        #endregion Public Operations

        #region Private Operations

        /// <summary>
        /// Пошук матриці, зворотної до "FLog", методом Жорданових виключень
        /// (Дана модифікація методу може використовуватися тільки в тих
випадках,
        /// коли (-a) = (a), тому що через непотрібність пропущена стадія зміни
елементів),
        /// крім того, відсутній пошук ненульового розв'язного елемента (у
випадку
        /// роботи з матрицею Вандермонда наявність нуля на діагоналі - збій
кодека,
        /// тому ситуація з виявленням нуля сприймається винятково як помилка
        /// </summary>
        /// <returns>Булевський прапор результату операції</returns>
        private bool FInv()
        {
            // Обчислюємо розподіл відсотків ітерацій по стадіях для
            // коректної обробки відсотків
            double allStageIter   = this.iterOfFirstStage +
this.iterOfSecondStage;
            int percOfFirstStage  = (int)((100.0 * this.iterOfFirstStage)  /
allStageIter);
            int percOfSecondStage = (int)((100.0 * this.iterOfSecondStage) /
allStageIter);

            // Дана стадія повинна займати хоча б один відсоток
            // (для коректності розрахунків)
            if (percOfSecondStage == 0)
            {
                percOfSecondStage = 1;
            }

            // Обчислюємо значення модуля, що дозволить виводити відсоток
обробки
            // рівно при одиничному збільшенні для циклу по "k"
            int progressMod1 = this.n / percOfSecondStage;

            // Якщо модуль дорівнює нулю, то збільшуємо його до значення "1",
щоб
            // прогрес виводився на кожній ітерації
            if (progressMod1 == 0)
            {
                progressMod1 = 1;
            }

            // Цикл вибору розв'язного елемента "pivot"
            for (int k = 0; k < this.n; ++k)
            {
                // Якщо даний рядок тривіальна - просто переходимо на нову
ітерацію
                if (this.FLogRowIsTrivial[k])
                {
                    continue;
                }

                // Зсув у масиві до елементів k-ой рядка
```

```
                int k_n = k * this.n;

                // Індекс розв'язного елемента
                int pivotIdx = k_n + k;

                // Витягаємо розв'язний елемент
                int pivot = this.FLog[pivotIdx];

                // Якщо розв'язний елемент дорівнює нулю - матриця не має
звороткої
                if (pivot == 0)
                {
                    //...указуємо на помилку конфігурації
                    this.configIsOK = false;

                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;

                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();

                    return false;
                }

                // Після добування розв'язного елемента поміщаємо на його місце
"1"
                this.FLog[pivotIdx] = 1;

                // Працюємо з рядками...
                for (int i = 0; i < this.n; i++)
                {
                    // Якщо перебуваємо на рядку розв'язного елемента -
переходимо
                    // на нову ітерацію
                    if (i == k)
                    {
                        continue;
                    }

                    // Зсув у масиві до елементів i-ой рядка
                    int i_n = i * this.n;

                    // Працюємо зі стовпцями...
                    for (int j = 0; j < this.n; j++)
                    {
                        // Якщо перебуваємо на стовпці розв'язного елемента -
переходимо
                        // на нову ітерацію...
                        if (j == k)
                        {
                            continue;
                        }

                        int idx = i_n + j;

                        //...а інакше робимо необхідні дії над матрицею:
                        // "A[i,j] = A[i,j] * pivot + A[i,k] * A[k,j]"
                        this.FLog[idx] = this.eGF16.Mul(this.FLog[idx], pivot) ^
this.eGF16.Mul(this.FLog[i_n + k], this.FLog[k_n + j]);
                    }
                }

                // Розподіл матриці на розв'язний елемент заміняємо множенням на
зворотний
                int pivotInv = this.eGF16.Inv(pivot);

                for (int i = 0; i < this.n; i++)
                {
                    // Зсув у масиві до елементів i-ой рядка
```

```
                    int i_n = (i * this.n);

                    for (int j = 0; j < this.n; j++)
                    {
                        int idx = i_n + j;

                        this.FLog[idx] = this.eGF16.Mul(this.FLog[idx],
pivotInv);
                    }
                }

                // Якщо є передплата на делегата відновлення прогресу -...
                if (
                        ((k % progressMod1) == 0)
                     &&
                        (OnUpdateCicleCodeMatrixFormingProgress != null)
                   )
                {
                    //...виводимо дані
                    OnUpdateCicleCodeMatrixFormingProgress((((double)(k + 1) /
(double)this.n) * percOfSecondStage) + percOfFirstStage);
                }

                // У випадку, якщо потрібна постановка на паузу, подію
"executeEvent"
                // буде скинуто, і будемо на паузі аж до його появи
                ManualResetEvent.WaitAll(this.executeEvent);

                // Якщо зазначено, що потрібно вийти з потоку - виходимо
                if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                {
                    // Указуємо, що декодер зконфігуровано некоректно
                    this.configIsOK = false;

                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;

                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();

                    return false;
                }
            }

            return true;
        }

        /// <summary>
        /// Обчислення логарифмів значень інвертованої матриці
        /// <summary>
        private void LogFCalc()
        {
            // Працюємо з рядками...
            for (int i = 0; i < this.n; i++)
            {
                // Зсув у масиві до елементів і-ой рядка
                int i_n = i * this.n;

                // Працюємо зі стовпцями...
                for (int j = 0; j < this.n; j++)
                {
                    int idx = i_n + j;

                    this.FLog[idx] = this.eGF16.Log(this.FLog[idx]);
                }
            }
        }

        /// <summary>
```

```
        /// Заповнення матриці "FLog" (матриці декодера) даними
        /// </summary>
        protected override void FillFLog()
        {
            // Якщо довжина вектора наявних томів менше кількості,
            // необхідного для відновлення...
            if (this.volList.Length < this.n)
            {
                //...указуємо на помилку конфігурації
                this.configIsOK = false;

                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();

                return;
            }

            // Виділяємо пам'ять під матрицю "FLog"
            this.FLog = new int[this.n * this.n];

            // Вектор лічильників всіх томів...
            int[] allVolCount = new int[this.n + this.m];

            //...і вектор ecc-томів для "затикання" пробілів, створених
            // загубленими основними томами
            int[] eccVolToFix = new int[this.m];

            // Лічильник кількості стертих основних томів
            int dataVolMissCount = this.n;

            // Ініціалізуємо масив лічильників всіх томів
            for (int i = 0; i < (this.n + this.m); i++)
            {
                allVolCount[i] = 0;
            }

            // Проводимо аналіз складу представлених томів на предмет наявності
основних
            for (int i = 0; i < this.n; i++)
            {
                // Обчислюємо номер поточного тому
                int currVol = Math.Abs(this.volList[i]);

                // Якщо номер тому відповідає припустимому діапазону
                if (currVol < (this.n + this.m))
                {
                    ++allVolCount[currVol];

                    // Якщо поточний том є основним, фіксуємо даний факт
                    if (currVol < this.n)
                    {
                        ---idataVolMissCount;
                    }

                } else
                {
                    // Указуємо на помилку конфігурації
                    this.configIsOK = false;

                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;

                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();

                    return;
```

```
                }
            }

            // Перевіряємо лічильники томів на помилкове дублювання
            for (int i = 0; i < (this.n + this.m); i++)
            {
                // Якщо деякий том був зазначений більш ніж один раз...
                if (allVolCount[i] > 1)
                {
                    //...указуємо на помилку конфігурації
                    this.configIsOK = false;

                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;

                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();

                    return;
                }
            }

            // Якщо перевірка на несуперечність не виявила проблем, починаємо
            // формувати матрицю "FLog"

            // Якщо основна конфігурація змінилася...
            if (this.mainConfigChanged)
            {
                if (this.eCicleCodeType == (int)CicleCodeType.Dispersal)
                {
                    //...робимо формування дисперсної матриці "D"
                    if (!MakeDispersalMatrix())
                    {
                        // Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;

                        // Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;

                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();

                        return;
                    }

                } else
                {
                    //...робимо формування альтернативного заповнення матриці
"A"
                    if (!MakeAlternativeMatrix())
                    {
                        // Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;

                        // Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;

                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();

                        return;
                    }
                }

                //...і скидаємо прапор
                this.mainConfigChanged = false;
            }
```

```
            // Для кожного загубленого основного тому шукаємо том для
відновлення
            for (int i = 0, j = 0; i < dataVolMissCount; i++)
            {
                // Рухаємося за списком томів доти, поки не знайдемо том для
                // відновлення для затикання "дірки" (основні томи мають номера
                // менше this.n (при нумерації з нуля!))
                while (this.volList[j] < this.n)
                {
                    j++;
                }

                // Зберігаємо номер тому для заміни загубленого основного тому
                eccVolToFix[i] = this.volList[j];

                j++; // j++ дозволяє перейти до наступного пошуку
            }

            // Працюємо по рядках матриці (в ідеалі, всі рядки повинні
заповнюватися
            // рядками з одиницею на головній діагоналі, що відповідає
відсутності
            // ушкоджень, але allVolCount укаже, якими є справи з наявністю
томів)
            for (int i = 0, e = 0; i < this.n; i++)
            {
                // Індекс рядка з дисперсної матриці, що буде поміщена в матрицю
кодування
                int DRowIdx;

                // Зсув у масиві до елементів i-ой рядка
                int i_n = i * this.n;

                // Якщо основний том відсутній, формуємо рядок матриці
Вандермонда
                if (allVolCount[i] == 0)
                {
                    // Обчислюємо номер рядка матриці Вандермонда, яку потрібно
вставити
                    // на місце даного рядка формованої матриці "FLog"
                    DRowIdx = eccVolToFix[e++];

                    // Указуємо, що даний рядок матриці "FLog" не тривіальна
                    this.FLogRowIsTrivial[i] = false;

                } else
                {
                    // Формуємо в матриці "FLog" нульовий рядок з одиницею на
головній діагоналі
                    // (відповідає наявному основному той)
                    DRowIdx = i;

                    // Указуємо, що даний рядок матриці "FLog" тривіальна
                    this.FLogRowIsTrivial[i] = true;
                }

                // Залежно від типу декодера беремо дані з відповідного масиву
                // (у ньому втримуються як рядки матриці Вандермонда, так і
"тривіальні" рядки,
                //  утримуючі нулі і єдиний елемент "1" на головній діагоналі)
                if (this.eCicleCodeType == (int)CicleCodeType.Dispersal)
                {
                    int bs = DRowIdx * this.n;

                    // Формування рядка в матриці кодування
                    // ("тривіальні" рядки вже втримуються в матриці "D", вони
вийшли
                    // "автоматично" на попередньому етапі обробки
MakeDispersal())
```

```
                    for (int j = 0; j < this.n; j++)
                    {
                        this.FLog[i_n + j] = this.D[bs + j];
                    }

                } else
                {
                    // Якщо це потрібно - формуємо "тривіальну" рядок...
                    if (this.FLogRowIsTrivial[i])
                    {
                        for (int j = 0; j < this.n; j++)
                        {
                            this.FLog[i_n + j] = 0;
                        }

                        this.FLog[i_n + i] = 1;

                    } else
                    {
                        int bs = (DRowIdx - this.n) * this.n;

                        //...а, інакше, беремо рядок матриці Вандермонда
                        for (int j = 0; j < this.n; j++)
                        {
                            this.FLog[i_n + j] = this.A[bs + j];
                        }
                    }
                }

                // У випадку, якщо потрібна постановка на паузу, подію
"executeEvent"
                // буде скинуто, і будемо на паузі аж до його появи
                ManualResetEvent.WaitAll(this.executeEvent);

                // Якщо зазначено, що потрібно вийти з потоку - виходимо
                if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                {
                    //...указуємо на помилку конфігурації
                    this.configIsOK = false;

                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;

                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();

                    return;
                }
            }

            // Знаходимо зворотну матрицю для "FLog"
            if (!FInv())
            {
                // Указуємо, що кодер зконфігуровано некоректно
                this.configIsOK = false;

                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();

                return;
            }

            // Обчислюємо логарифми елементів інвертованої матриці
            LogFCalc();

            // Якщо є передплата на делегата завершення...
```

```
            if (OnCicleCodeMatrixFormingFinish != null)
            {
                //...повідомляємо, що екземпляр класу готовий до роботи
                OnCicleCodeMatrixFormingFinish();
            }

            // Активуємо індикатор актуального стану змінних-членів
            this.finished = true;

            // Установлюємо подію завершення обробки
            this.finishedEvent[0].Set();
        }

        #endregion Private Operations

        #region Public Properties

        /// <summary>
        /// Список порядкових номерів наявних томів
        /// </summary>
        public int[] VolList
        {
            get
            {
                if (!InProcessing)
                {
                    return this.volList;

                } else
                {
                    return null;
                }
            }
        }

        #endregion Public Properties
    }
}
```

**Файл CicleCodeEncoder.cs – кодер циклічного коду для підвищення надійності функціонування ЦОД**

```
using System;
using System.Threading;

namespace RecoveryDisk
{
    /// <summary>
    /// Клас кодера циклічного коду для підвищення надійності функціонування ЦОД
    /// </summary>
    public class CicleCodeEncoder : CicleCodeBase
    {
        #region Construction & Destruction

        /// <summary>
        /// Конструктор кодера за замовчуванням
        /// </summary>
        public CicleCodeEncoder()
        {
            // Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }

        /// <summary>
        /// Базовий конструктор кодера
        /// </summary>
        /// <param name="dataCount">Кількість основних томів</param>
        /// <param name="eccCount">Кількість томів для відновлення</param>
        public CicleCodeEncoder(int dataCount, int eccCount)
        {
            // Установка конфігурації кодера
            SetConfig(dataCount, eccCount, (int)CicleCodeType.Alternative);

            // Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }

        /// <summary>
        /// Розширений конструктор кодера
        /// </summary>
        /// <param name="dataCount">Кількість основних томів</param>
        /// <param name="eccCount">Кількість томів для відновлення</param>
        /// <param name="codecType">Тип кодера циклічного коду для підвищення
надійності функціонування ЦОД (по типу матриці)</param>
        public CicleCodeEncoder(int dataCount, int eccCount, int codecType)
        {
            // Установка конфігурації кодера
            SetConfig(dataCount, eccCount, codecType);

            // Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }

        #endregion Construction & Destruction

        #region Public Operations

        /// <summary>
        /// Установка конфігурації кодера
        /// </summary>
        /// <param name="dataCount">Кількість основних томів</param>
        /// <param name="eccCount">Кількість томів для відновлення</param>
        /// <param name="codecType">Тип кодека кодера циклічного коду для
підвищення надійності функціонування ЦОД (по типу матриці)</param>
        /// <returns>Булевський прапор операції установки конфігурації</returns>
```

```
        public bool SetConfig(int dataCount, int eccCount, int codecType)
        {
            int maxVolCount;

            // Установлюємо константи, що відповідають обраному режиму
            if (codecType == (int)CicleCodeType.Dispersal)
            {
                maxVolCount = (int)CicleCodeConst.MaxVolCountDisp;

            } else
            {
                maxVolCount = (int)CicleCodeConst.MaxVolCountAlt;
            }

            // Перевіряємо конфігурацію на коректність
            if (
                    (dataCount > 0)
                 &&
                    (eccCount > 0)
                 &&
                    ((dataCount + eccCount) <= maxVolCount)
                )
            {
                // Якщо основна конфігурація змінилася - сповіщаємо про це
                if (
                        (dataCount != this.n)
                     ||
                        (eccCount != this.m)
                     ||
                        (codecType != this.eCicleCodeType)
                    )
                {
                    this.mainConfigChanged = true;
                }

                // Зберігаємо конфігурацію
                this.n = dataCount;
                this.m = eccCount;
                this.eCicleCodeType = codecType;

                // Також перераховуємо кількість ітерацій всіх стадій підготовки
                double n = this.n;
                double m = this.m;

                // Нормалізуємо значення для розрахунку, щоб уникнути
переповнення змінних
                NormalizeNM(ref n, ref m);

                // Кількість ітерацій на першій стадії залежить від типу
використовуваної матриці
                if (this.eCicleCodeType == (int)CicleCodeType.Alternative)
                {
                    this.iterOfFirstStage = m;

                } else
                {
                    this.iterOfFirstStage = ((n * m) * n) + (n * ((n + m) + (n *
(n + m))));
                }

                this.iterOfSecondStage = 0; // У кодери немає інвертування
матриці

                this.configIsOK = true;

            } else
            {
                //...указуємо на помилку конфігурації
                this.configIsOK = false;
```

```
            }

            return this.configIsOK;
        }

        /// <summary>
        /// Метод множення матриці кодування на вхідний прологарифмований вектор
        /// </summary>
        /// <param name="dataLog">Прологарифмований вхідний вектор (вихідні
дані)</param>
        /// <param name="ecc">Вихідний вектор (надлишкові дані)</param>
        /// <returns>Булевський прапор результату операції</returns>
        public bool Process(int[] dataLog, ref int[] ecc)
        {
            // Якщо кодер зконфігуровано некоректно, обробка неможлива!
            if (!this.configIsOK)
            {
                return false;
            }

            // Копіюємо покажчик на масив експонент для скорочення часу обігу
            int[] GF16Exp = this.eGF16.GFExpTable;

            // Обчислення результату множення матриці на вектор
            for (int i = 0; i < this.m; i++)
            {
                int mulSum = 0;         // Сума добутку рядка матриці на
стовпець
                int i_n = i * this.n;   // Зсув у масиві до елементів i-ой рядка

                for (int j = 0; j < this.n; j++)
                {
                    mulSum ^= GF16Exp[this.FLog[i_n + j] + dataLog[j]];
                }

                ecc[i] = mulSum;
            }

            return true;
        }

        #endregion Public Operations

        #region Private Operations

        /// <summary>
        /// Заповнення матриці Вандермонда даними
        /// </summary>
        protected override void FillFLog()
        {
            // Якщо основна конфігурація змінилася...
            if (this.mainConfigChanged)
            {
                if (this.eCicleCodeType == (int)CicleCodeType.Dispersal)
                {
                    //...робимо формування дисперсної матриці "D"
                    if (!MakeDispersalMatrix())
                    {
                        // Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;

                        // Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;

                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();

                        return;
                    }
```

```
                } else
                {
                    //...робимо формування альтернативного заповнення матриці
"A"
                    if (!MakeAlternativeMatrix())
                    {
                        // Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;

                        // Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;

                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();

                        return;
                    }
                }

                // Виділяємо пам'ять під матрицю "FLog"
                this.FLog = new int[this.m * this.n];

                // Заповнюємо матрицю кодування
                for (int i = 0; i < this.m; i++)
                {
                    // Зсув у масиві до елементів i-ой рядка
                    int i_n = i * this.n;

                    // Залежно від типу кодера беремо дані з відповідного масиву
                    if (this.eCicleCodeType == (int)CicleCodeType.Dispersal)
                    {
                        // Формування рядка в матриці кодування
                        for (int j = 0; j < this.n; j++)
                        {
                            // У матрицю кодування поміщаємо логарифми її
вихідних елементів
                            // (для прискорення множення матриці на вектор)
                            this.FLog[i_n + j] = this.eGF16.Log(this.D[((this.n
+ i) * this.n) + j]);
                        }

                    } else
                    {
                        // Формування рядка в матриці кодування
                        for (int j = 0; j < this.n; j++)
                        {
                            int idx = i_n + j;

                            // У матрицю кодування поміщаємо логарифми її
вихідних елементів
                            // (для прискорення множення матриці на вектор)
                            this.FLog[idx] = this.eGF16.Log(this.A[idx]);
                        }
                    }

                    // У випадку, якщо потрібна постановка на паузу, подію
"executeEvent"
                    // буде скинуто, і будемо на паузі аж до його появи
                    ManualResetEvent.WaitAll(this.executeEvent);

                    // Якщо зазначено, що потрібно вийти з потоку - виходимо
                    if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                    {
                        // Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;

                        // Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;
```

```
                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();

                        return;
                    }
                }

                // Якщо є передплата на делегата завершення...
                if (OnCicleCodeMatrixFormingFinish != null)
                {
                    //...повідомляємо, що екземпляр класу готовий до роботи
                    OnCicleCodeMatrixFormingFinish();
                }

                //...і скидаємо прапор
                this.mainConfigChanged = false;
            }

            // Якщо є передплата на делегата завершення...
            if (OnCicleCodeMatrixFormingFinish != null)
            {
                //...повідомляємо, що екземпляр класу готовий до роботи
                OnCicleCodeMatrixFormingFinish();
            }

            // Активуємо індикатор актуального стану змінних-членів
            this.finished = true;

            // Установлюємо подію завершення обробки
            this.finishedEvent[0].Set();
        }

        #endregion Private Operations
    }
}
```

**Файл ProcessForm.cs - вікно відображення процесів запису/читання та перевірки цілісності даних**

```
namespace RecoveryDisk
{
    partial class ProcessForm
    {
        /// <summary>
                /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true якщо керуючі ресурси повинні бути
розташовані, у іншому випадку  false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Необхідний метод для підтримки розробника - не модифікується
        /// зміст цього метода використовується редактором коду.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(ProcessForm));
            this.processPriorityGroupBox = new System.Windows.Forms.GroupBox();
            this.processPriorityComboBox = new System.Windows.Forms.ComboBox();
            this.processGroupBox = new System.Windows.Forms.GroupBox();
            this.processProgressBar = new System.Windows.Forms.ProgressBar();
            this.fileAnalyzeStatGroupBox = new System.Windows.Forms.GroupBox();
            this.percOfAltEccLabel = new System.Windows.Forms.Label();
            this.percOfDamageLabel = new System.Windows.Forms.Label();
            this.percOfAltEccLabel_ = new System.Windows.Forms.Label();
            this.percOfDamageLabel_ = new System.Windows.Forms.Label();
            this.logGroupBox = new System.Windows.Forms.GroupBox();
            this.logListBox = new System.Windows.Forms.ListBox();
            this.countGroupBox = new System.Windows.Forms.GroupBox();
            this.errorCountLabel = new System.Windows.Forms.Label();
            this.okCountLabel = new System.Windows.Forms.Label();
            this.errorPictureBox = new System.Windows.Forms.PictureBox();
            this.okPictureBox = new System.Windows.Forms.PictureBox();
            this.errorCountLabel_ = new System.Windows.Forms.Label();
            this.okCountLabel_ = new System.Windows.Forms.Label();
            this.toolTip = new System.Windows.Forms.ToolTip(this.components);
            this.stopButtonXP = new PinkieControls.ButtonXP();
            this.pauseButtonXP = new PinkieControls.ButtonXP();
            this.closingTimer = new System.Windows.Forms.Timer(this.components);
            this.processTimer = new System.Windows.Forms.Timer(this.components);
            this.processPriorityGroupBox.SuspendLayout();
            this.processGroupBox.SuspendLayout();
            this.fileAnalyzeStatGroupBox.SuspendLayout();
            this.logGroupBox.SuspendLayout();
            this.countGroupBox.SuspendLayout();

((System.ComponentModel.ISupportInitialize)(this.errorPictureBox)).BeginInit();
```

```
((System.ComponentModel.ISupportInitialize)(this.okPictureBox)).BeginInit();
            this.SuspendLayout();
            //
            // processPriorityGroupBox
            //

this.processPriorityGroupBox.Controls.Add(this.processPriorityComboBox);
            this.processPriorityGroupBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.processPriorityGroupBox.Location = new
System.Drawing.Point(617, 216);
            this.processPriorityGroupBox.Name = "processPriorityGroupBox";
            this.processPriorityGroupBox.Size = new System.Drawing.Size(135,
64);
            this.processPriorityGroupBox.TabIndex = 0;
            this.processPriorityGroupBox.TabStop = false;
            this.processPriorityGroupBox.Text = "Пріоритет процесу";
            //
            // processPriorityComboBox
            //
            this.processPriorityComboBox.BackColor =
System.Drawing.SystemColors.Control;
            this.processPriorityComboBox.DropDownStyle =
System.Windows.Forms.ComboBoxStyle.DropDownList;
            this.processPriorityComboBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.processPriorityComboBox.FormattingEnabled = true;
            this.processPriorityComboBox.Items.AddRange(new object[] {
            "За замовчуванням",
            "Знижений",
            "Нормальний",
            "Підвищений",
            "Найвищий"});
            this.processPriorityComboBox.Location = new System.Drawing.Point(9,
33);
            this.processPriorityComboBox.Name = "processPriorityComboBox";
            this.processPriorityComboBox.Size = new System.Drawing.Size(117,
21);
            this.processPriorityComboBox.TabIndex = 0;
            this.processPriorityComboBox.TabStop = false;
            this.toolTip.SetToolTip(this.processPriorityComboBox, "Список
можливих значень пріоритету процесу обробки");
            this.processPriorityComboBox.SelectedIndexChanged += new
System.EventHandler(this.processPriorityComboBox_SelectedIndexChanged);
            //
            // processGroupBox
            //
            this.processGroupBox.Controls.Add(this.processProgressBar);
            this.processGroupBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.processGroupBox.Location = new System.Drawing.Point(12, 9);
            this.processGroupBox.Name = "processGroupBox";
            this.processGroupBox.Size = new System.Drawing.Size(871, 65);
            this.processGroupBox.TabIndex = 0;
            this.processGroupBox.TabStop = false;
            this.processGroupBox.Text = "Обробка";
            //
            // processProgressBar
            //
            this.processProgressBar.Location = new System.Drawing.Point(14, 30);
            this.processProgressBar.Name = "processProgressBar";
            this.processProgressBar.Size = new System.Drawing.Size(844, 20);
            this.processProgressBar.Style =
System.Windows.Forms.ProgressBarStyle.Continuous;
            this.processProgressBar.TabIndex = 0;
            //
            // fileAnalyzeStatGroupBox
            //
```

```
            this.fileAnalyzeStatGroupBox.Controls.Add(this.percOfAltEccLabel);
            this.fileAnalyzeStatGroupBox.Controls.Add(this.percOfDamageLabel);
            this.fileAnalyzeStatGroupBox.Controls.Add(this.percOfAltEccLabel_);
            this.fileAnalyzeStatGroupBox.Controls.Add(this.percOfDamageLabel_);
            this.fileAnalyzeStatGroupBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.fileAnalyzeStatGroupBox.Location = new System.Drawing.Point(12,
216);
            this.fileAnalyzeStatGroupBox.Name = "fileAnalyzeStatGroupBox";
            this.fileAnalyzeStatGroupBox.Size = new System.Drawing.Size(459,
64);
            this.fileAnalyzeStatGroupBox.TabIndex = 0;
            this.fileAnalyzeStatGroupBox.TabStop = false;
            this.fileAnalyzeStatGroupBox.Text = "Результат аналізу цілісності
даних";
            //
            // percOfAltEccLabel
            //
            this.percOfAltEccLabel.AutoSize = true;
            this.percOfAltEccLabel.Location = new System.Drawing.Point(195, 41);
            this.percOfAltEccLabel.Name = "percOfAltEccLabel";
            this.percOfAltEccLabel.Size = new System.Drawing.Size(10, 13);
            this.percOfAltEccLabel.TabIndex = 0;
            this.percOfAltEccLabel.Text = "-";
            //
            // percOfDamageLabel
            //
            this.percOfDamageLabel.AutoSize = true;
            this.percOfDamageLabel.Location = new System.Drawing.Point(195, 20);
            this.percOfDamageLabel.Name = "percOfDamageLabel";
            this.percOfDamageLabel.Size = new System.Drawing.Size(10, 13);
            this.percOfDamageLabel.TabIndex = 0;
            this.percOfDamageLabel.Text = "-";
            //
            // percOfAltEccLabel_
            //
            this.percOfAltEccLabel_.AutoSize = true;
            this.percOfAltEccLabel_.Location = new System.Drawing.Point(7, 41);
            this.percOfAltEccLabel_.Name = "percOfAltEccLabel_";
            this.percOfAltEccLabel_.Size = new System.Drawing.Size(163, 13);
            this.percOfAltEccLabel_.TabIndex = 0;
            this.percOfAltEccLabel_.Text = "Резерв перевірочних даних для
відновлення:";
            //
            // percOfDamageLabel_
            //
            this.percOfDamageLabel_.AutoSize = true;
            this.percOfDamageLabel_.Location = new System.Drawing.Point(7, 20);
            this.percOfDamageLabel_.Name = "percOfDamageLabel_";
            this.percOfDamageLabel_.Size = new System.Drawing.Size(148, 13);
            this.percOfDamageLabel_.TabIndex = 0;
            this.percOfDamageLabel_.Text = "Всього пошкоджених секторів:";
            //
            // logGroupBox
            //
            this.logGroupBox.Controls.Add(this.logListBox);
            this.logGroupBox.Location = new System.Drawing.Point(12, 80);
            this.logGroupBox.Name = "logGroupBox";
            this.logGroupBox.Size = new System.Drawing.Size(871, 130);
            this.logGroupBox.TabIndex = 0;
            this.logGroupBox.TabStop = false;
            this.logGroupBox.Text = "Лог процесу";
            //
            // logListBox
            //
            this.logListBox.BackColor = System.Drawing.SystemColors.Control;
            this.logListBox.BorderStyle = System.Windows.Forms.BorderStyle.None;
            this.logListBox.FormattingEnabled = true;
            this.logListBox.HorizontalScrollbar = true;
```

```
            this.logListBox.Location = new System.Drawing.Point(7, 23);
            this.logListBox.Name = "logListBox";
            this.logListBox.SelectionMode =
System.Windows.Forms.SelectionMode.None;
            this.logListBox.Size = new System.Drawing.Size(851, 91);
            this.logListBox.TabIndex = 0;
            this.logListBox.TabStop = false;
            this.logListBox.UseTabStops = false;
            this.logListBox.SelectedIndexChanged += new
System.EventHandler(this.logListBox_SelectedIndexChanged);
            //
            // countGroupBox
            //
            this.countGroupBox.Controls.Add(this.errorCountLabel);
            this.countGroupBox.Controls.Add(this.okCountLabel);
            this.countGroupBox.Controls.Add(this.errorPictureBox);
            this.countGroupBox.Controls.Add(this.okPictureBox);
            this.countGroupBox.Controls.Add(this.errorCountLabel_);
            this.countGroupBox.Controls.Add(this.okCountLabel_);
            this.countGroupBox.Location = new System.Drawing.Point(482, 216);
            this.countGroupBox.Name = "countGroupBox";
            this.countGroupBox.Size = new System.Drawing.Size(124, 64);
            this.countGroupBox.TabIndex = 0;
            this.countGroupBox.TabStop = false;
            this.countGroupBox.Text = "Лічільник процесу";
            //
            // errorCountLabel
            //
            this.errorCountLabel.AutoSize = true;
            this.errorCountLabel.Location = new System.Drawing.Point(63, 41);
            this.errorCountLabel.Name = "errorCountLabel";
            this.errorCountLabel.Size = new System.Drawing.Size(13, 13);
            this.errorCountLabel.TabIndex = 0;
            this.errorCountLabel.Text = "0";
            this.toolTip.SetToolTip(this.errorCountLabel, "Лічильник некоректно
оброблених файлів");
            //
            // okCountLabel
            //
            this.okCountLabel.AutoSize = true;
            this.okCountLabel.Location = new System.Drawing.Point(63, 20);
            this.okCountLabel.Name = "okCountLabel";
            this.okCountLabel.Size = new System.Drawing.Size(13, 13);
            this.okCountLabel.TabIndex = 0;
            this.okCountLabel.Text = "0";
            this.toolTip.SetToolTip(this.okCountLabel, "Лічильник коректно
оброблених файлів");
            //
            // errorPictureBox
            //
            this.errorPictureBox.Image =
((System.Drawing.Image)(resources.GetObject("errorPictureBox.Image")));
            this.errorPictureBox.Location = new System.Drawing.Point(10, 40);
            this.errorPictureBox.Name = "errorPictureBox";
            this.errorPictureBox.Size = new System.Drawing.Size(21, 15);
            this.errorPictureBox.TabIndex = 2;
            this.errorPictureBox.TabStop = false;
            //
            // okPictureBox
            //
            this.okPictureBox.Image =
((System.Drawing.Image)(resources.GetObject("okPictureBox.Image")));
            this.okPictureBox.Location = new System.Drawing.Point(10, 19);
            this.okPictureBox.Name = "okPictureBox";
            this.okPictureBox.Size = new System.Drawing.Size(21, 15);
            this.okPictureBox.TabIndex = 1;
            this.okPictureBox.TabStop = false;
            //
            // errorCountLabel_
```

```
            //
            this.errorCountLabel_.AutoSize = true;
            this.errorCountLabel_.Location = new System.Drawing.Point(28, 41);
            this.errorCountLabel_.Name = "errorCountLabel_";
            this.errorCountLabel_.Size = new System.Drawing.Size(35, 13);
            this.errorCountLabel_.TabIndex = 0;
            this.errorCountLabel_.Text = "Error :";
            this.toolTip.SetToolTip(this.errorCountLabel_, "Лічильник некоректно
оброблених файлів");
            //
            // okCountLabel_
            //
            this.okCountLabel_.AutoSize = true;
            this.okCountLabel_.Location = new System.Drawing.Point(28, 20);
            this.okCountLabel_.Name = "okCountLabel_";
            this.okCountLabel_.Size = new System.Drawing.Size(28, 13);
            this.okCountLabel_.TabIndex = 0;
            this.okCountLabel_.Text = "OK :";
            this.toolTip.SetToolTip(this.okCountLabel_, "Лічильник коректно
оброблених файлів");
            //
            // toolTip
            //
            this.toolTip.AutomaticDelay = 2000;
            this.toolTip.AutoPopDelay = 20000;
            this.toolTip.InitialDelay = 2000;
            this.toolTip.ReshowDelay = 1000;
            //
            // stopButtonXP
            //
            this.stopButtonXP.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(0)))), ((int)(((byte)(236)))),
((int)(((byte)(233)))), ((int)(((byte)(216)))));
            this.stopButtonXP.DefaultScheme = true;
            this.stopButtonXP.DialogResult =
System.Windows.Forms.DialogResult.None;
            this.stopButtonXP.Hint = "";
            this.stopButtonXP.Location = new System.Drawing.Point(762, 257);
            this.stopButtonXP.Name = "stopButtonXP";
            this.stopButtonXP.Scheme = PinkieControls.ButtonXP.Schemes.Blue;
            this.stopButtonXP.Size = new System.Drawing.Size(121, 23);
            this.stopButtonXP.TabIndex = 2;
            this.stopButtonXP.Text = "Перервати обробку";
            this.toolTip.SetToolTip(this.stopButtonXP, "Припинення обробки
файлів із закриттям даного вікна");
            this.stopButtonXP.Click += new
System.EventHandler(this.stopButtonXP_Click);
            //
            // pauseButtonXP
            //
            this.pauseButtonXP.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(0)))), ((int)(((byte)(236)))),
((int)(((byte)(233)))), ((int)(((byte)(216)))));
            this.pauseButtonXP.DefaultScheme = true;
            this.pauseButtonXP.DialogResult =
System.Windows.Forms.DialogResult.None;
            this.pauseButtonXP.Hint = "";
            this.pauseButtonXP.Location = new System.Drawing.Point(762, 220);
            this.pauseButtonXP.Name = "pauseButtonXP";
            this.pauseButtonXP.Scheme = PinkieControls.ButtonXP.Schemes.Blue;
            this.pauseButtonXP.Size = new System.Drawing.Size(121, 23);
            this.pauseButtonXP.TabIndex = 1;
            this.pauseButtonXP.Text = "Пауза";
            this.toolTip.SetToolTip(this.pauseButtonXP, "Постановка/зняття
процесу обробки з паузи");
            this.pauseButtonXP.Click += new
System.EventHandler(this.pauseButtonXP_Click);
            //
            // closingTimer
```

```
            //
            this.closingTimer.Tick += new
System.EventHandler(this.closingTimer_Tick);
            //
            // processTimer
            //
            this.processTimer.Interval = 500;
            this.processTimer.Tick += new
System.EventHandler(this.processTimer_Tick);
            //
            // ProcessForm
            //
            this.AutoScaleDimensions = new System.Drawing.Size(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(894, 292);
            this.ControlBox = false;
            this.Controls.Add(this.stopButtonXP);
            this.Controls.Add(this.pauseButtonXP);
            this.Controls.Add(this.countGroupBox);
            this.Controls.Add(this.processPriorityGroupBox);
            this.Controls.Add(this.logGroupBox);
            this.Controls.Add(this.fileAnalyzeStatGroupBox);
            this.Controls.Add(this.processGroupBox);
            this.DoubleBuffered = true;
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.Icon =
((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
            this.MaximizeBox = false;
            this.MinimizeBox = false;
            this.Name = "ProcessForm";
            this.ShowInTaskbar = false;
            this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterScreen;
            this.Text = "Обробка файлу";
            this.Load += new System.EventHandler(this.ProcessForm_Load);
            this.processPriorityGroupBox.ResumeLayout(false);
            this.processGroupBox.ResumeLayout(false);
            this.fileAnalyzeStatGroupBox.ResumeLayout(false);
            this.fileAnalyzeStatGroupBox.PerformLayout();
            this.logGroupBox.ResumeLayout(false);
            this.countGroupBox.ResumeLayout(false);
            this.countGroupBox.PerformLayout();

((System.ComponentModel.ISupportInitialize)(this.errorPictureBox)).EndInit();

((System.ComponentModel.ISupportInitialize)(this.okPictureBox)).EndInit();
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.GroupBox processPriorityGroupBox;
        private System.Windows.Forms.GroupBox processGroupBox;
        private System.Windows.Forms.ProgressBar processProgressBar;
        private System.Windows.Forms.GroupBox fileAnalyzeStatGroupBox;
        private System.Windows.Forms.Label percOfDamageLabel_;
        private System.Windows.Forms.Label percOfAltEccLabel_;
        private System.Windows.Forms.GroupBox logGroupBox;
        private System.Windows.Forms.GroupBox countGroupBox;
        private System.Windows.Forms.Label errorCountLabel_;
        private System.Windows.Forms.Label okCountLabel_;
        private System.Windows.Forms.ListBox logListBox;
        private System.Windows.Forms.ComboBox processPriorityComboBox;
        private System.Windows.Forms.PictureBox errorPictureBox;
        private System.Windows.Forms.PictureBox okPictureBox;
        private System.Windows.Forms.Label errorCountLabel;
        private System.Windows.Forms.Label okCountLabel;
```

```
        private System.Windows.Forms.ToolTip toolTip;
        private System.Windows.Forms.Timer closingTimer;
        private System.Windows.Forms.Label percOfAltEccLabel;
        private System.Windows.Forms.Label percOfDamageLabel;
        private PinkieControls.ButtonXP pauseButtonXP;
        private PinkieControls.ButtonXP stopButtonXP;
        private System.Windows.Forms.Timer processTimer;
    }
}
```

**Файл FileAnalyzer.cs - контроль цілісності даних**

```
using System;
using System.Threading;
using System.IO;

namespace RecoveryDisk
{
    /// <summary>
    /// Клас контролю цілісності набору файлів-томів
    /// </summary>
    public class FileAnalyzer
    {
        #region Delegates

        /// <summary>
        /// Делегат відновлення прогресу контролю цілісності файлів
        /// </summary>
        public OnUpdateDoubleValueHandler OnUpdateFileAnalyzeProgress;

        /// <summary>
        /// Делегат завершення процесу контролю цілісності файлів
        /// </summary>
        public OnEventHandler OnFileAnalyzeFinish;

        /// <summary>
        /// Делегат одержання статистики ушкоджень багатотомного архіву
        /// </summary>
        public OnUpdateTwoDoubleValueHandler OnGetDamageStat;

        #endregion Delegates

        #region Public Properties & Data

        /// <summary>
        /// Булевська властивість "Файл обробляється?"
        /// </summary>
        public bool InProcessing
        {
            get
            {
                if (
                        (this.thrFileAnalyzer != null)
                     &&
                        (
                              (this.thrFileAnalyzer.ThreadState ==
ThreadState.Running)
                           ||
                              (this.thrFileAnalyzer.ThreadState ==
ThreadState.WaitSleepJoin)
                        )
                    )
                {
                    return true;

                } else
                {
                    return false;
                }
            }
        }

        /// <summary>
        /// Булевська властивість "Екземпляр класу закінчив обробку
        /// (має актуальний стан змінних-членів)?"
        /// </summary>
```

```
public bool Finished
{
    get
    {
        // Якщо клас не зайнятий обробкою - повертаємо значення
        if (!InProcessing)
        {
            return this.finished;

        } else
        {
            return false;
        }
    }
}

/// <summary>
/// Екземпляр класу повністю закінчив обробку?
/// </summary>
private bool finished;

/// <summary>
/// Булевська властивість "Безліч файлів оброблена коректно?"
/// </summary>
public bool ProcessedOK
{
    get
    {
        // Якщо клас не зайнятий обробкою - повертаємо значення
        if (!InProcessing)
        {
            return this.processedOK;

        } else
        {
            return false;
        }
    }
}

/// <summary>
/// Обробка набору файлів зроблена коректно?
/// </summary>
private bool processedOK;

/// <summary>
/// Список порядкових номерів наявних томів
/// </summary>
public int[] VolList
{
    get
    {
        if (!InProcessing)
        {
            return this.volList;

        } else
        {
            return null;
        }
    }
}

/// <summary>
/// Вектор, що вказує на состав томів
/// </summary>
private int[] volList;

/// <summary>
```

```
        /// Всі томи для відновлення коректні?
        /// </summary>
        public bool AllEccVolsOK
        {
            get
            {
                if (!InProcessing)
                {
                    return this.allEccVolsOK;

                } else
                {
                    return false;
                }
            }
        }

        /// <summary>
        /// Всі томи для відновлення коректні?
        /// </summary>
        private bool allEccVolsOK;

        /// <summary>
        /// Пріоритет процесу
        /// </summary>
        public int ThreadPriority
        {
            get
            {
                return (int)this.threadPriority;
            }

            set
            {
                if (
                        (this.thrFileAnalyzer != null)
                     &&
                        (this.thrFileAnalyzer.IsAlive)
                    )
                {
                    switch (value)
                    {
                        default:
                        case 0:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.Lowest;

                            break;
                        }

                        case 1:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.BelowNormal;

                            break;
                        }

                        case 2:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.Normal;

                            break;
                        }

                        case 3:
                        {
```

```
                            this.threadPriority =
System.Threading.ThreadPriority.AboveNormal;

                            break;
                        }

                        case 4:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.Highest;

                            break;
                        }
                    }

                    // Установлюємо обраний пріоритет процесу
                    this.thrFileAnalyzer.Priority = this.threadPriority;

                    // Дублюємо установку параметра для підконтрольного об'єкта
                    if (this.eFileIntegrityCheck != null)
                    {
                        this.eFileIntegrityCheck.ThreadPriority = value;
                    }
                }
            }
        }

        /// <summary>
        /// Пріоритет процесу контролю цілісності файлів
        /// </summary>
        private ThreadPriority threadPriority;

        /// <summary>
        /// Подія, установлювана по завершенню обробки
        /// </summary>
        public ManualResetEvent[] FinishedEvent
        {
            get
            {
                return this.finishedEvent;
            }
        }

        /// <summary>
        /// Подія, установлювана по завершенню обробки
        /// </summary>
        private ManualResetEvent[] finishedEvent;

        #endregion Public Properties & Data

        #region Data

        /// <summary>
        /// Модуль для впакування (розпакування) ім'я файлу в префіксний формат
        /// </summary>
        private FileNamer eFileNamer;

        /// <summary>
        /// Екземпляр класу контролю цілісності набору файлів
        /// </summary>
        private FileIntegrityCheck eFileIntegrityCheck;

        /// <summary>
        /// Шлях до файлів для обробки
        /// </summary>
        private String path;

        /// <summary>
        /// Ім'я файлу, якому належить безліч томів
```

```
        /// </summary>
        private String fileName;

        /// <summary>
        /// Кількість основних томів
        /// </summary>
        private int dataCount;

        /// <summary>
        /// Кількість томів для відновлення
        /// </summary>
        private int eccCount;

        /// <summary>
        /// Тип кодера циклічного коду для підвищення надійності функціонування
ЦОД (по типу використовуваної матриці кодування)
        /// </summary>
        private int codecType;

        /// <summary>
        /// Використовується швидке добування з томів (без перевірки CRC-64)?
        /// </summary>
        private bool fastExtraction;

        /// <summary>
        /// Потік контролю цілісності файлу
        /// </summary>
        private Thread thrFileAnalyzer;

        /// <summary>
        /// Подія припинення обробки файлів
        /// </summary>
        private ManualResetEvent[] exitEvent;

        /// <summary>
        /// Подія продовження обробки файлів
        /// </summary>
        private ManualResetEvent[] executeEvent;

        /// <summary>
        /// Подія "пробудження" циклу очікування
        /// </summary>
        private ManualResetEvent[] wakeUpEvent;

        #endregion Data

        #region Construction & Destruction

        /// <summary>
        /// Конструктор класу перевірки цілісності набору файлів
        /// </summary>
        public FileAnalyzer()
        {
            // Модуль для впакування (розпакування) ім'я файлу в префіксний
формат
            this.eFileNamer = new FileNamer();

            // Створюємо екземпляр класу контролю цілісності набору файлів
            this.eFileIntegrityCheck = new FileIntegrityCheck();

            // Шлях до файлів для обробки за замовчуванням порожній
            this.path = "";

            // Ініціалізуємо ім'я файлу за замовчуванням
            this.fileName = "NONAME";

            // Спочатку всі томи для відновлення вважаємо ушкодженими
            this.allEccVolsOK = false;
```

```
            // Екземпляр класу повністю закінчив обробку?
            this.finished = true;

            // Обробка зроблена коректно?
            this.processedOK = false;

            // За замовчуванням встановлюється фоновий пріоритет
            this.threadPriority = 0;

            // Ініціалізуємо подію припинення обробки файлів
            this.exitEvent = new ManualResetEvent[] { new
ManualResetEvent(false) };

            // Ініціалізуємо подію продовження обробки файлів
            this.executeEvent = new ManualResetEvent[] { new
ManualResetEvent(false) };

            // Ініціалізуємо подію "пробудження" циклу очікування
            this.wakeUpEvent = new ManualResetEvent[] { new
ManualResetEvent(false) };

            // Подія, установлювана по завершенню обробки
            this.finishedEvent = new ManualResetEvent[] { new
ManualResetEvent(true) };
        }

        #endregion Construction & Destruction

        #region Public Operations

        /// <summary>
        /// Метод запуску потоку обробки обчислення й записи CRC64 у кінець
файлів
        /// </summary>
        /// <param name="path">Шлях до файлів для обробки</param>
        /// <param name="fileName">Ім'я файлу для обробки</param>
        /// <param name="dataCount">Конфігурація кількості основних
томів</param>
        /// <param name="eccCount">Конфігурація кількості томів для
відновлення</param>
        /// <param name="codecType">Тип кодера циклічного коду для підвищення
надійності функціонування ЦОД (по типу матриці)</param>
        /// <param name="runAsSeparateThread">Запускати в окремому
потоці?</param>
        /// <returns>Булевський прапор операції</returns>
        public bool StartToWriteCRC64(String path, String fileName, int
dataCount, int eccCount, int codecType, bool runAsSeparateThread)
        {
            // Якщо потік обчислення CRC-64 працює - не дозволяємо повторний
запуск
            if (InProcessing)
            {
                return false;
            }

            // Скидаємо прапор коректності результату перед запуском потоку
            this.processedOK = false;

            // Скидаємо індикатор актуального стану змінних-членів
            this.finished = false;

            // Зберігаємо шлях до файлів для обробки
            if (path == null)
            {
                this.path = "";

            } else
            {
                // Робимо виділення шляху з "path" у випадку,
```

```
                // якщо туди було записано повне ім'я
                this.path = this.eFileNamer.GetPath(path);
            }

            if (fileName == null)
            {
                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();

                return false;
            }

            // Робимо виділення короткого ім'я файлу з "fileName" у випадку,
            // якщо туди було записано повне ім'я
            this.fileName = this.eFileNamer.GetShortFileName(fileName);

            // Перевіряємо на некоректну конфігурацію
            if (
                    (dataCount <= 0)
                 ||
                    (eccCount <= 0)
                 ||
                    ((dataCount + eccCount) >
(int)CicleCodeConst.MaxVolCountAlt)
                )
            {
                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();

                return false;
            }

            // Зберігаємо кількість основних томів
            this.dataCount = dataCount;

            // Зберігаємо кількість томів для відновлення
            this.eccCount = eccCount;

            // Зберігаємо тип кодера циклічного коду для підвищення надійності
функціонування ЦОД (по типу використовуваної матриці кодування)
            this.codecType = codecType;

            // Указуємо, що потік повинен виконуватися
            this.exitEvent[0].Reset();
            this.executeEvent[0].Set();
            this.wakeUpEvent[0].Reset();
            this.finishedEvent[0].Reset();

            // Якщо зазначено, що не потрібен запуск в окремому потоці,
            // запускаємо в даному
            if (!runAsSeparateThread)
            {
                // Обчислюємо CRC-64 для кожного з файлів набору
                WriteCRC64();

                // Повертаємо результат обробки
                return this.processedOK;
            }

            // Створюємо потік обчислення й запису CRC-64...
            this.thrFileAnalyzer = new Thread(new ThreadStart(WriteCRC64));

            //...потім даємо йому ім'я...
```

```
            this.thrFileAnalyzer.Name = "FileAnalyzer.WriteCRC64()";

            //...установлюємо обраний пріоритет завдання...
            this.thrFileAnalyzer.Priority = this.threadPriority;

            //...і запускаємо його
            this.thrFileAnalyzer.Start();

            // Повідомляємо, що все нормально
            return true;
        }

        /// <summary>
        /// Метод запуску потоку обробки перевірки CRC64, записаного в кінець
        /// кожного з файлів набору, з генеруванням списку наявних томів
"volList",
        /// який буде використаний декодером для відновлення даних
        /// </summary>
        /// <param name="path">Шлях до файлів для обробки</param>
        /// <param name="fileName">Ім'я файлу для обробки</param>
        /// <param name="dataCount">Конфігурація кількості основних
томів</param>
        /// <param name="eccCount">Конфігурація кількості томів для
відновлення</param>
        /// <param name="codecType">Тип кодера циклічного коду для підвищення
надійності функціонування ЦОД (по типу матриці)</param>
        /// <param name="fastExtraction">Використовується швидке добування з
томів (без перевірки CRC-64)?</param>
        /// <param name="runAsSeparateThread">Запускати в окремому
потоці?</param>
        /// <returns>Булевський прапор операції</returns>
        public bool StartToAnalyzeCRC64(String path, String fileName, int
dataCount, int eccCount, int codecType, bool fastExtraction, bool
runAsSeparateThread)
        {
            // Якщо потік обчислення CRC-64 працює - не дозволяємо повторний
запуск
            if (InProcessing)
            {
                return false;
            }

            // Спочатку всі томи для відновлення вважаємо ушкодженими
            this.allEccVolsOK = false;

            // Скидаємо прапор коректності результату перед запуском потоку
            this.processedOK = false;

            // Скидаємо індикатор актуального стану змінних-членів
            this.finished = false;

            // Зберігаємо шлях до файлів для обробки
            if (path == null)
            {
                this.path = "";

            } else
            {
                // Робимо виділення шляху з "path" у випадку,
                // якщо туди було записано повне ім'я
                this.path = this.eFileNamer.GetPath(path);
            }

            if (fileName == null)
            {
                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
```

```
                this.finishedEvent[0].Set();

                return false;
            }

            // Робимо виділення короткого ім'я файлу з "fileName" у випадку,
            // якщо туди було записано повне ім'я
            this.fileName = this.eFileNamer.GetShortFileName(fileName);

            // Перевіряємо на некоректну конфігурацію
            if (
                    (dataCount <= 0)
                 ||
                    (eccCount <= 0)
                 ||
                    ((dataCount + eccCount) >
(int)CicleCodeConst.MaxVolCountAlt)
                )
            {
                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();

                return false;
            }

            // Зберігаємо кількість основних томів
            this.dataCount = dataCount;

            // Зберігаємо кількість томів для відновлення
            this.eccCount = eccCount;

            // Зберігаємо тип кодека кодера циклічного коду для підвищення
надійності функціонування ЦОД (по типу використовуваної матриці кодування)
            this.codecType = codecType;

            // Використовується швидке добування з томів (без перевірки CRC-64)?
            this.fastExtraction = fastExtraction;

            // Указуємо, що потік повинен виконуватися
            this.exitEvent[0].Reset();
            this.executeEvent[0].Set();
            this.wakeUpEvent[0].Reset();
            this.finishedEvent[0].Reset();

            // Якщо зазначено, що не потрібен запуск в окремому потоці,
            // запускаємо в даному
            if (!runAsSeparateThread)
            {
                // Обчислюємо й перевіряємо CRC-64 для кожного з файлів набору
із заповненням
                // властивості VolList
                AnalyzeCRC64();

                // Повертаємо результат обробки
                return this.processedOK;
            }

            // Створюємо потік обчислення й перевірки CRC-64...
            this.thrFileAnalyzer = new Thread(new ThreadStart(AnalyzeCRC64));

            //...потім даємо йому ім'я...
            this.thrFileAnalyzer.Name = "FileAnalyzer.AnalyzeCRC64()";

            //...установлюємо обраний пріоритет завдання...
            this.thrFileAnalyzer.Priority = this.threadPriority;
```

```
            //...і запускаємо його
            this.thrFileAnalyzer.Start();

            // Повідомляємо, що все нормально
            return true;
        }

        /// <summary>
        /// Метод зупинки потоку
        /// </summary>
        public void Stop()
        {
            // Указуємо, що потік обробки більше не повинен виконуватися
            this.exitEvent[0].Set();

            // Примусово знімаємо з паузи
            this.executeEvent[0].Set();

            // Знімаємо з очікування в циклі
            this.wakeUpEvent[0].Set();
        }

        /// <summary>
        /// Постановка потоку обробки на паузу
        /// </summary>
        public void Pause()
        {
            // Ставимо на паузу
            this.executeEvent[0].Reset();

            // Знімаємо з очікування в циклі
            this.wakeUpEvent[0].Set();
        }

        /// <summary>
        /// Зняття потоку обробки з паузи
        /// </summary>
        public void Continue()
        {
            // Знімаємо обробку с паузи
            this.executeEvent[0].Set();
        }

        #endregion Public Operations

        #region Private Operations

        /// <summary>
        /// Обчислення й запис у кінець файлів значення CRC-64
        /// </summary>
        private void WriteCRC64()
        {
            // Обчислюємо значення модуля, що дозволить виводити відсоток
обробки
            // рівно при одиничному збільшенні для циклу по "i"
            int progressMod1 = (this.dataCount + this.eccCount) / 100;

            // Якщо модуль дорівнює нулю, то збільшуємо його до значення "1",
щоб
            // прогрес виводився на кожній ітерації (файл дуже маленький)
            if (progressMod1 == 0)
            {
                progressMod1 = 1;
            }

            // Піддаємо обробці всі томи
            for (int volNum = 0; volNum < (this.dataCount + this.eccCount);
volNum++)
            {
```

```
                // Зчитуємо первісне ім'я файлу
                String fileName = this.fileName;

                // Одержуємо ім'я вихідного файлу в префіксній формі
                this.eFileNamer.Pack(ref fileName, volNum, this.dataCount,
this.eccCount, this.codecType);

                // Формуємо повне ім'я файлу
                fileName = this.path + fileName;

                // Робимо обчислення CRC-64 для кожного файлу
                if (this.eFileIntegrityCheck.StartToWriteCRC64(fileName, true))
                {
                    // Цикл очікування завершення обробки файлу
                    while (true)
                    {
                        // Якщо не виявили встановленої події "executeEvent",
                        // те користувач хоче, щоб ми поставили обробку на паузу
-
                        if (!ManualResetEvent.WaitAll(this.executeEvent, 0,
false))
                        {
                            //...припиняємо роботу контрольованого алгоритму...
                            this.eFileIntegrityCheck.Pause();

                            //...програма переходить у режим сна
                            ManualResetEvent.WaitAll(this.executeEvent);

                            // А коли прокинулися, указуємо, що обробка повинна
тривати
                            this.eFileIntegrityCheck.Continue();
                        }

                        // Чекаємо кожне з перерахованих подій...
                        int eventIdx = ManualResetEvent.WaitAny(new
ManualResetEvent[] { this.wakeUpEvent[0], this.exitEvent[0],
this.eFileIntegrityCheck.FinishedEvent[0] });

                        //...якщо одержали сигнал до того, щоб прокинутися -
                        // переходимо на нову ітерацію, тому що прокидаємося
                        // перед постановкою на паузу...
                        if (eventIdx == 0)
                        {
                            //...попередньо скинувши подію, що змусила нас
прокинутися
                            this.wakeUpEvent[0].Reset();

                            continue;
                        }

                        //...якщо одержали сигнал до виходу з обробки...
                        if (eventIdx == 1)
                        {
                            //...зупиняємо контрольований алгоритм
                            this.eFileIntegrityCheck.Stop();

                            // Указуємо на те, що обробка була перервана
                            this.processedOK = false;

                            // Активуємо індикатор актуального стану змінних-
членів
                            this.finished = true;

                            // Установлюємо подію завершення обробки
                            this.finishedEvent[0].Set();

                            return;
                        }
```

```
                        //...якщо одержали сигнал про завершення обробки
вкладеним алгоритмом...
                        if (eventIdx == 2)
                        {
                            //...виходимо із циклу очікування завершення (цього
й чекали в while(true)!)
                            break;
                        }

                    } // while(true)

                } else
                {
                    // Скидаємо прапор коректності результату
                    this.processedOK = false;

                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;

                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();

                    return;
                }

                // У зв'язку із закриттям великої кількості файлових потоків
                // необхідно дочекатися запису змін, внесених потоком
                // кодування в тіло класу. Потік уже не працює, але
                // установлена їм Булевська властивість, можливо, ще
                // "не виявилося"
                for (int i = 0; i < (int)WaitCount.MaxWaitCount; i++)
                {
                    if (!this.eFileIntegrityCheck.Finished)
                    {
                        Thread.Sleep((int)WaitTime.MinWaitTime);

                    } else
                    {
                        break;
                    }
                }

                // Якщо цикли очікування закриття файлових потоків не привели до
бажаного
                // результату - це помилка
                if (!this.eFileIntegrityCheck.ProcessedOK)
                {
                    // Указуємо на те, що обробка не була завершена коректно
                    this.processedOK = false;

                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;

                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();

                    return;
                }

                // Виводимо прогрес обробки
                if (
                        ((volNum % progressMod1) == 0)
                     &&
                        (OnUpdateFileAnalyzeProgress != null)
                    )
                {
                    OnUpdateFileAnalyzeProgress(((double)(volNum + 1) /
(double)(this.dataCount + this.eccCount)) * 100.0);
                }
```

```
                // У випадку, якщо потрібна постановка на паузу, подію
"executeEvent"
                // буде скинуто, і будемо на паузі аж до його появи
                ManualResetEvent.WaitAll(this.executeEvent);

                // Якщо зазначено, що потрібно вийти з потоку - виходимо
                if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                {
                    // Указуємо на те, що обробка була перервана
                    this.processedOK = false;

                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;

                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();

                    return;
                }
            }

            // Повідомляємо про закінчення процесу обробки
            if (OnFileAnalyzeFinish != null)
            {
                OnFileAnalyzeFinish();
            }

            // Повідомляємо, що обробка пройшла коректно
            this.processedOK = true;

            // Активуємо індикатор актуального стану змінних-членів
            this.finished = true;

            // Установлюємо подію завершення обробки
            this.finishedEvent[0].Set();
        }

          /// <summary>
        /// Обчислення й перевірка значення CRC-64, записаного наприкінці файлу
        /// </summary>
        private void AnalyzeCRC64()
        {
            // Обчислюємо значення модуля, що дозволить виводити відсоток
обробки
            // рівно при одиничному збільшенні для циклу по "i"
            int progressMod1 = (this.dataCount + this.eccCount) / 100;

            // Якщо модуль дорівнює нулю, то збільшуємо його до значення "1",
щоб
            // прогрес виводився на кожній ітерації (файл дуже маленький)
            if (progressMod1 == 0)
            {
                progressMod1 = 1;
            }

            // Виділяємо пам'ять під "volList"
            this.volList = new int[this.dataCount];

            // Виділяємо пам'ять під "altEccList"
            int[] altEccList = new int[this.eccCount];

            // Індекс у масиві томів
            int volListIdx = 0;

            // Індекс у масиві томів для відновлення
            int altEccListIdx = 0;

            // Лічильник кількості ушкоджених основних томів
```

```
            int dataVolMissCount = 0;

            // Лічильник кількості знайдених томів для відновлення
            int eccVolPresentCount = 0;

            // Ім'я файлу для обробки
            String fileName;

            // Піддаємо перевірці всі основні томи
            for (int dataNum = 0; dataNum < this.dataCount; dataNum++)
            {
                // Спочатку припускаємо, що поточний том ушкоджено
                bool dataVolIsOK = false;

                // Зчитуємо первісне ім'я файлу
                fileName = this.fileName;

                // Одержуємо ім'я вихідного файлу в префіксній формі
                this.eFileNamer.Pack(ref fileName, dataNum, this.dataCount,
this.eccCount, this.codecType);

                // Формуємо повне ім'я файлу
                fileName = this.path + fileName;

                // Якщо вихідний файл існує...
                if (File.Exists(fileName))
                {
                    // Якщо не використовується швидке добування - перевіряємо
на цілісність
                    // CRC-64, інакше думаємо, що все корректно (цілісність тому
беремо
                    // по факті його наявності)
                    if (!this.fastExtraction)
                    {
                        //...- робимо його перевірку
                        if (this.eFileIntegrityCheck.StartToCheckCRC64(fileName,
true))
                        {
                            // Цикл очікування завершення обробки файлу
                            while (true)
                            {
                                // Якщо не виявили встановленої події
"executeEvent",
                                // те користувач хоче, щоб ми поставили обробку
на паузу -
                                if (!ManualResetEvent.WaitAll(this.executeEvent,
0, false))
                                {
                                    //...припиняємо роботу контрольованого
алгоритму...
                                    this.eFileIntegrityCheck.Pause();

                                    //...програма переходить у режим сна
                                    ManualResetEvent.WaitAll(this.executeEvent);

                                    // А коли прокинулися, указуємо, що обробка
повинна тривати
                                    this.eFileIntegrityCheck.Continue();
                                }

                                // Чекаємо кожне з перерахованих подій...
                                int eventIdx = ManualResetEvent.WaitAny(new
ManualResetEvent[] { this.wakeUpEvent[0], this.exitEvent[0],
this.eFileIntegrityCheck.FinishedEvent[0] });

                                //...якщо одержали сигнал до того, щоб
прокинутися -
                                // переходимо на нову ітерацію, тому що
прокидаємося
```

```
                                // перед постановкою на паузу...
                                if (eventIdx == 0)
                                {
                                    //...попередньо скинувши подію, що змусила
нас прокинутися
                                    this.wakeUpEvent[0].Reset();

                                    continue;
                                }

                                //...якщо одержали сигнал до виходу з обробки...
                                if (eventIdx == 1)
                                {
                                    //...зупиняємо контрольований алгоритм
                                    this.eFileIntegrityCheck.Stop();

                                    // Указуємо на те, що обробка була перервана
                                    this.processedOK = false;

                                    // Активуємо індикатор актуального стану
змінних-членів
                                    this.finished = true;

                                    // Установлюємо подію завершення обробки
                                    this.finishedEvent[0].Set();

                                    return;
                                }

                                //...якщо одержали сигнал про завершення обробки
вкладеним алгоритмом...
                                if (eventIdx == 2)
                                {
                                    //...виходимо із циклу очікування завершення
(цього й чекали в while(true)!)
                                    break;
                                }

                            } // while(true)

                        } else
                        {
                            // Скидаємо прапор коректності результату
                            this.processedOK = false;

                            // Активуємо індикатор актуального стану змінних-
членів
                            this.finished = true;

                            // Установлюємо подію завершення обробки
                            this.finishedEvent[0].Set();

                            return;
                        }

                        // У зв'язку із закриттям великої кількості файлових
потоків
                        // необхідно дочекатися запису змін, внесених потоком
                        // кодування в тіло класу. Потік уже не працює, але
                        // установлене їм Булевська властивість, можливо, ще
                        // "не виявилося"
                        for (int i = 0; i < (int)WaitCount.MaxWaitCount; i++)
                        {
                            if (!this.eFileIntegrityCheck.Finished)
                            {
                                Thread.Sleep((int)WaitTime.MinWaitTime);

                            } else
                            {
```

```
                                break;
                            }
                        }

                        // Указуємо, що основний том коректний
                        if (this.eFileIntegrityCheck.ProcessedOK)
                        {
                            dataVolIsOK = true;
                        }

                    } else
                    {
                        // Указуємо, що основний том коректний
                        dataVolIsOK = true;
                    }

                    // Виводимо прогрес обробки
                    if (
                            ((dataNum % progressMod1) == 0)
                     &&
                            (OnUpdateFileAnalyzeProgress != null)
                        )
                    {
                        OnUpdateFileAnalyzeProgress(((double)(dataNum + 1) /
(double)(this.dataCount + this.eccCount)) * 100.0);
                    }

                    // У випадку, якщо потрібна постановка на паузу, подію
"executeEvent"
                    // буде скинуто, і будемо на паузі аж до його появи
                    ManualResetEvent.WaitAll(this.executeEvent);

                    // Якщо зазначено, що потрібно вийти з потоку - виходимо
                    if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                    {
                        // Указуємо на те, що обробка була перервана
                        this.processedOK = false;

                        // Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;

                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();

                        return;
                    }
                }

                // Якщо даний основний том не ушкоджений, записуємо його в
"volList",
                // а інакше збільшуємо лічильник ушкоджених томів і ставимо на
місце
                // номера тому значення "-1", що вкаже на необхідність
підстановки
                // тому для відновлення
                if (dataVolIsOK)
                {
                    this.volList[volListIdx++] = dataNum;

                } else
                {
                    this.volList[volListIdx++] = -1;

                    // Збільшуємо лічильник кількості ушкоджених основних томів
                    dataVolMissCount++;
                }
            }

            // Тепер, коли знаємо кількість ушкоджених основних томів,
```

```
            // потрібно просканувати всі файли для відновлення, і визначити
            // необхідну їхню частину в список томів, а "надлишок" помістити в
            // список альтренативних томів для відновлення
            for (int eccNum = this.dataCount; eccNum < (this.dataCount +
this.eccCount); eccNum++)
            {
                // Спочатку припускаємо, що поточний том ушкоджено
                bool eccVolIsOK = false;

                // Зчитуємо первісне ім'я файлу
                fileName = this.fileName;

                // Одержуємо ім'я вихідного файлу в префіксній формі
                this.eFileNamer.Pack(ref fileName, eccNum, this.dataCount,
this.eccCount, this.codecType);

                // Формуємо повне ім'я файлу
                fileName = this.path + fileName;

                // Якщо вихідний файл існує...
                if (File.Exists(fileName))
                {
                    // Якщо не використовується швидке добування - перевіряємо
на цілісність
                    // CRC-64, інакше думаємо, що все корректно (цілісність тому
беремо
                    // по факту його наявності)
                    if (!this.fastExtraction)
                    {
                        //...- робимо його перевірку
                        if (this.eFileIntegrityCheck.StartToCheckCRC64(fileName,
true))
                        {
                            // Цикл очікування завершення обробки файлу
                            while (true)
                            {
                                // Якщо не виявили встановленої події
"executeEvent",
                                // то користувач хоче, щоб ми поставили обробку
на паузу -
                                if (!ManualResetEvent.WaitAll(this.executeEvent,
0, false))
                                {
                                    //...припиняємо роботу контрольованого
алгоритму...
                                    this.eFileIntegrityCheck.Pause();

                                    //...програма переходить у режим сна
                                    ManualResetEvent.WaitAll(this.executeEvent);

                                    // А коли прокинулися, указуємо, що обробка
повинна тривати
                                    this.eFileIntegrityCheck.Continue();
                                }

                                // Чекаємо кожне з перерахованих подій...
                                int eventIdx = ManualResetEvent.WaitAny(new
ManualResetEvent[] { this.wakeUpEvent[0], this.exitEvent[0],
this.eFileIntegrityCheck.FinishedEvent[0] });

                                //...якщо одержали сигнал до того, щоб
прокинутися -
                                // переходимо на нову ітерацію, тому що
прокидаємося
                                // перед постановкою на паузу...
                                if (eventIdx == 0)
                                {
                                    //...попередньо скинувши подію, що змусила
нас прокинутися
```

```
                                this.wakeUpEvent[0].Reset();

                                continue;
                            }

                            //...якщо одержали сигнал до виходу з обробки...
                            if (eventIdx == 1)
                            {
                                //...зупиняємо контрольований алгоритм
                                this.eFileIntegrityCheck.Stop();

                                // Указуємо на те, що обробка була перервана
                                this.processedOK = false;

                                // Активуємо індикатор актуального стану
змінних-членів
                                this.finished = true;

                                // Установлюємо подію завершення обробки
                                this.finishedEvent[0].Set();

                                return;
                            }

                            //...якщо одержали сигнал про завершення обробки
вкладеним алгоритмом...
                            if (eventIdx == 2)
                            {
                                //...виходимо із циклу очікування завершення
(цього й чекали в while(true)!)
                                break;
                            }

                        } // while(true)

                    } else
                    {
                        // Скидаємо прапор коректності результату
                        this.processedOK = false;

                        // Активуємо індикатор актуального стану змінних-
членів
                        this.finished = true;

                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();

                        return;
                    }

                    // У зв'язку із закриттям великої кількості файлових
потоків
                    // необхідно дочекатися запису змін, внесених потоком
                    // кодування в тіло класу. Потік уже не працює, але
                    // установлена їм Булевська властивість, можливо, ще
                    // "не виявилася"
                    for (int i = 0; i < (int)WaitCount.MaxWaitCount; i++)
                    {
                        if (!this.eFileIntegrityCheck.Finished)
                        {
                            Thread.Sleep((int)WaitTime.MinWaitTime);

                        } else
                        {
                            break;
                        }
                    }

                    // Указуємо, що том для відновлення коректний
```

```
                        if (this.eFileIntegrityCheck.ProcessedOK)
                        {
                            eccVolIsOK = true;
                        }

                    } else
                    {
                        // Указуємо, що том для відновлення коректний
                        eccVolIsOK = true;
                    }

                    // Виводимо прогрес обробки
                    if (
                            ((eccNum % progressMod1) == 0)
                         &&
                            (OnUpdateFileAnalyzeProgress != null)
                        )
                    {
                        OnUpdateFileAnalyzeProgress(((double)(eccNum + 1) /
(double)(this.dataCount + this.eccCount)) * 100.0);
                    }

                    // У випадку, якщо потрібна постановка на паузу, подію
"executeEvent"
                    // буде скинуто, і будемо на паузі аж до його появи
                    ManualResetEvent.WaitAll(this.executeEvent);

                    // Якщо зазначено, що потрібно вийти з потоку - виходимо
                    if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                    {
                        // Указуємо на те, що обробка була перервана
                        this.processedOK = false;

                        // Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;

                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();

                        return;
                    }
                }

                // Якщо том для відновлення гарний...
                if (eccVolIsOK)
                {
                    //...- додаємо його в список
                    altEccList[altEccListIdx++] = eccNum;

                    // Збільшуємо лічильник кількості томів для відновлення
                    eccVolPresentCount++;

                } else
                {
                    //...а інакше вказуємо, що том ушкоджено
                    altEccList[altEccListIdx++] = -1;
                }
            }

            // Якщо значення лічильника кількості коректних томів для
відновлення збігається
            // зі значенням лічильника томів для відновлення конфігурації - всі
томи для
            // відновлення є неушкодженими
            if (eccVolPresentCount == this.eccCount)
            {
                this.allEccVolsOK = true;
            }
```

```
            // Виводимо статистику ушкоджень
            if (OnGetDamageStat != null)
            {
                // Обчислюємо загальний відсоток ушкоджень (суму ушкоджень
                // основних томів і томів для відновлення ділимо на загальну
кількість томів)
                double percOfDamage = ((double)(dataVolMissCount +
(this.eccCount - eccVolPresentCount)) / (double)(this.dataCount +
this.eccCount)) * 100;

                // Обчислюємо відсоток "" альтернативних томів, щовижили, для
відновлення
                // Альтернативні томи - це спочатку ті томи, які не планується
використовувати для відновлення
                double percOfAltEcc = ((double)(eccVolPresentCount -
dataVolMissCount) / (double)this.eccCount) * 100;

                // Виводимо статистику ушкоджень
                OnGetDamageStat(percOfDamage, percOfAltEcc);
            }

            // Якщо немає ушкоджених основних томів, просто виходимо
            if (dataVolMissCount == 0)
            {
                // Повідомляємо про закінчення процесу обробки
                if (OnFileAnalyzeFinish != null)
                {
                    OnFileAnalyzeFinish();
                }

                // Указуємо на те, що дані не ушкоджені
                this.processedOK = true;

                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();

                return;
            }

            // Якщо ми не зможемо відновити ушкодження...
            if (eccVolPresentCount < dataVolMissCount)
            {
                //...указуємо на те, що дані не можуть бути відновлені
                this.processedOK = false;

                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();

                return;
            }

            // Переміщаємося на початок списку альтернативних томів для
відновлення
            altEccListIdx = 0;

            // Тепер пробігаємося по вектору "volList", і замість кожного зі
значень "-1"
            // підставляємо чергове значення зі знайденого діапазону
            for (int i = 0; i < this.dataCount; i++)
            {
                if (this.volList[i] == -1)
                {
                    // Пробігаємося по векторі томів для відновлення,
```

```
                        // зупиняючись на коректному томі для відновлення
                        while (altEccList[altEccListIdx] == -1)
                        {
                            altEccListIdx++;
                        }

                        // Підставляємо на місце ушкодженого основного тому
                        // том для відновлення,...
                        this.volList[i] = altEccList[altEccListIdx];

                        //...забираючи використаний том зі списку альтернативних
                        altEccList[altEccListIdx] = -1;
                    }
                }

                // Повідомляємо про закінчення процесу обробки
                if (OnFileAnalyzeFinish != null)
                {
                    OnFileAnalyzeFinish();
                }

                // Повідомляємо, що обробка пройшла коректно
                this.processedOK = true;

                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();
            }

            #endregion Private Operations
        }
    }
```

**Файл About.cs - вікно довідки про програму**

```
namespace RecoveryData
{
    partial class AboutForm
    {
        /// <summary>
        /// Необхідні змінні розробника.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true якщо керуючі ресурси повинні бути
розташовані, у іншому випадку  false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Необхідний метод для підтримки розробника - не модифікується
        /// зміст цього метода використовується редактором коду.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.developersListBoxdevelopersListBox = new
System.Windows.Forms.ListBox();
            this.CicleCodeIconTimer = new
System.Windows.Forms.Timer(this.components);
            this.okButtonXP = new PinkieControls.ButtonXP();
            this.SuspendLayout();
            //
            // developersListBoxdevelopersListBox
            //
            this.developersListBoxdevelopersListBox.BackColor =
System.Drawing.SystemColors.Control;
            this.developersListBoxdevelopersListBox.BorderStyle =
System.Windows.Forms.BorderStyle.None;
            this.developersListBoxdevelopersListBox.FormattingEnabled = true;
            this.developersListBoxdevelopersListBox.Items.AddRange(new object[]
{
            "БАКЛАВРСЬКА ДИПЛОМНА РОБОТА",
            "",
            "На тему:",
            "",
            "Програмне забезпечення системи підвищення надійності функціонування
ЦОД",
            "",
            "",
            "Керівник: Марченко К.М.",
            "",
            "Розробив: студент Голяга Денис Русланович ",
            "                  гр. КІ-21-2",
            "",
            "м. Кропивницький 2025"});
            this.developersListBoxdevelopersListBox.Location = new
System.Drawing.Point(11, 6);
            this.developersListBoxdevelopersListBox.Name =
"developersListBoxdevelopersListBox";
```

```
            this.developersListBoxdevelopersListBox.SelectionMode =
System.Windows.Forms.SelectionMode.None;
            this.developersListBoxdevelopersListBox.Size = new
System.Drawing.Size(330, 182);
            this.developersListBoxdevelopersListBox.TabIndex = 0;
            this.developersListBoxdevelopersListBox.TabStop = false;
            this.developersListBoxdevelopersListBox.SelectedIndexChanged += new
System.EventHandler(this.developersListBoxdevelopersListBox_SelectedIndexChanged
);
            //
            // CicleCodeIconTimer
            //
            this.CicleCodeIconTimer.Interval = 40;
            this.CicleCodeIconTimer.Tick += new
System.EventHandler(this.CicleCodeIconTimer_Tick);
            //
            // okButtonXP
            //
            this.okButtonXP.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(0)))), ((int)(((byte)(236)))),
((int)(((byte)(233)))), ((int)(((byte)(216)))));
            this.okButtonXP.DefaultScheme = true;
            this.okButtonXP.DialogResult =
System.Windows.Forms.DialogResult.None;
            this.okButtonXP.Hint = "";
            this.okButtonXP.Location = new System.Drawing.Point(266, 177);
            this.okButtonXP.Name = "okButtonXP";
            this.okButtonXP.Scheme = PinkieControls.ButtonXP.Schemes.Blue;
            this.okButtonXP.Size = new System.Drawing.Size(75, 23);
            this.okButtonXP.TabIndex = 0;
            this.okButtonXP.Text = "OK";
            this.okButtonXP.Click += new
System.EventHandler(this.okButtonXP_Click);
            //
            // AboutForm
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(350, 212);
            this.Controls.Add(this.okButtonXP);
            this.Controls.Add(this.developersListBoxdevelopersListBox);
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.MaximizeBox = false;
            this.MinimizeBox = false;
            this.Name = "AboutForm";
            this.ShowInTaskbar = false;
            this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterParent;
            this.Text = "Про програму...";
            this.Load += new System.EventHandler(this.AboutForm_Load);
            this.FormClosing += new
System.Windows.Forms.FormClosingEventHandler(this.AboutForm_FormClosing);
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.ListBox developersListBoxdevelopersListBox;
        private System.Windows.Forms.Timer CicleCodeIconTimer;
        private PinkieControls.ButtonXP okButtonXP;

    }
}
```