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
“____” _____ 2024 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення ієрархічних систем зберігання даних з використанням технології Tiered Storage”

Виконав здобувач вищої освіти
IV курсу, групи КІ-21-3СК
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Казаченко В.І.
«____» _____ 2024 р.

Керівник проекту
доктор філософії (PhD)
_____ Усік П.С
«____» _____ 2024 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 2024 року

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

Казаченку Владиславу Ігоровичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення ієрархічних систем зберігання даних з використанням технології Tiered Storage*

2. Керівник роботи *Усік Павло Сергійович, доктор філософії (PhD)*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 132-02 від 01.04.2024 року

3. Строк подання студентом роботи до захисту *23.05.2024 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення ієрархічних систем зберігання даних з використанням технології Tiered Storage*

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 2024 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	10.03.2024 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2024 р.	
3.	Розробка моделі компонента	20.03.2024 р.	
4.	Розробка структур даних	25.03.2024 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2024 р.	
6.	Програмування алгоритмів	10.04.2024 р.	
7.	Оформлення ПЗ	17.04.2024 р.	
8.	Попередній захист роботи	23.05.2024 р.	

Дата видачі завдання
« 17 » січня 2024 р.

Підпис керівника
_____ Усік П.С
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2024 р.

Підпис здобувача
_____ Казаченко В.І.
(прізвище та ініціали)

АНОТАЦІЯ

Казаченко В.І. Програмне забезпечення ієрархічних систем зберігання даних з використанням технології Tiered Storage. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2024.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для ієрархічних систем зберігання даних з використанням технології Tiered Storage.

Метою розробки є програмне забезпечення ієрархічних систем зберігання даних з використанням технології Tiered Storage.

Результат роботи – програмна реалізація ієрархічних систем зберігання даних з використанням технології Tiered Storage.

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 Delphi 10.4 Sydney.

Ключові слова: комп'ютерна інженерія, зберігання даних, Tiered Storage

ABSTRACT

Kazachenko V.I. Software for hierarchical data storage systems using Tiered Storage technology. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2024.

In this graduation thesis for the first (bachelor) level of higher education, software is developed, which is intended for hierarchical data storage systems using Tiered Storage technology.

The purpose of the development is software for hierarchical data storage systems using Tiered Storage technology.

The result of the work is the software implementation of hierarchical data storage systems using Tiered Storage technology.

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 software tools are provided.

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

The program was developed in the Delphi 10.4 Sydney environment.

Keywords: computer **engineering, data storage, Tiered Storage**

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

LZW – Алгоритм Зіва-Лемпела

ВСТУП

Актуальність теми. Потреба в зберіганні даних робить ринок систем зберігання даних (СЗД) найменш підданим волатильності продажів. Незважаючи на складну економічну ситуацію, компанії продовжують інвестувати в інфраструктуру зберігання. В Україні, по даним IDC, в III кварталі 2023 року виторг виробників зовнішніх систем зберігання даних досягла 119 млн доларів, що в річному вираженні відповідає росту на 19,2%. Сумарна ємність придбаних споживачами зовнішніх СЗД склала 65,8 тис. Тбайт – майже на 30% велике, ніж роком раніше. Трійка лідерів незмінна: EMC, HP і IBM спільно втримують близько 80% ринку (у грошовому вираженні). EMC зберігає першість завдяки успішним продажам продуктів різних категорій – від молодшого до старшого класу, а також систем для хмарного зберігання й роботи з Великими Даними.

Поряд зі зростаючими обсягами інформації, які генеруються в результаті транзакцій, інвестиціям у СЗД сприяють нові додатки, особливо хмарні сервіси. Усе велике широке поширення в Україні одержують технології віртуалізації СЗД, але великість компаній як і раніше воліють використовувати відособлене встаткування для різних робочих процесів, не створюючи єдиного пула ресурсів зберігання. Тим часом користувачам усе складніше орієнтуватися у величезних масивах неструктурованих даних, тому при роботі з контентом потрібні нові підходи. До систем зберігання, їх функціональних можливостей, продуктивності, масштабованості пред'являються високі вимоги.

Мета й завдання дослідження. Метою роботи є програмне забезпечення ієрархічних систем зберігання даних з використанням технології Tiered Storage.

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

– Огляд існуючих систем зберігання даних з використанням технології Tiered Storage.

– Дослідження ієрархічних систем зберігання даних з використанням технології Tiered Storage.

– Програмна реалізація ієрархічних систем зберігання даних з використанням технології Tiered Storage.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі зберігання даних з використанням технології Tiered Storage.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення ієрархічних систем зберігання даних з використанням технології Tiered Storage, 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

За інформацією IDC, за останні п'ять років вартість зберігання гігабайта інформації в корпоративних СЗД скоротилася більш ніж у сім разів, однак нових технологічних можливостей для її подальшого зниження колишніми темпами поки не передбачається.

Оскільки немає ніяких підстав уважати, що бюджети на IT у замовників помітно виростуть, керівники IT-підрозділів будуть змушені намагатися вирішити поставлені завдання з меншими витратами. Наприклад, замість покупки нових систем – модернізувати наявні, а замість систем старшого класу – здобувати СЗД середнього класу із близьким функціоналом. У результаті росте інтерес до недорогих рішень азіатських компаній: зсув попиту убік систем зберігання початкового рівня продовжиться. Однак, як прогнозує IDC, одночасно буде збільшуватися обсяг продажів старших моделей СЗД середнього класу й систем високого рівня, які знайдуть застосування в цілому ряді проектів.

Оптимізація операційних витрат при організації зберігання даних змусить обертати велике уваги на комплексну ефективність проектів, оптимізувати використання ресурсів і зважувати витрати на інтеграцію рішень. На зміну екстенсивному нарощуванню обсягів даних приходять нові технології й підходи: віртуалізація СЗД, дедуплікація, багаторівневе зберігання інформації, аутсорсинг зберігання даних, хмарне зберігання (Cloud Storage). Виробники прагнуть змістити пріоритети: концепція розвитку інфраструктури усе велике перемикається з екстенсивної на інтенсивну модель, однак в IDC уважають, що ці зусилля поки не зустрічають помітного відгуку в російських замовників, оскільки вартість передових інструментів керування, оптимізації й дедуплікації залишається досить високої.

1.2 Область застосування

Дуже строгі вимоги до СЗД по надійності й продуктивності пред'являє серверна віртуалізація. Важлива тенденція – спільне використання віртуалізованих систем зберігання, серверів і мереж, однак така конвергенція тільки починає проникати на ринок.

Подальше поширення й розвиток одержать твердотільні накопичувачі (Solid State Drive, SSD), СЗД у проектах віртуалізації, віртуальне/динамічне виділення ресурсів (Thin Provisioning), віддалена реплікація й засоби автоматизації. Бажання компаній оптимізувати використання своїх ресурсів стимулює попит на вдосконалене керуюче ПЗ, засоби централізованого адміністрування й керування, відстеження рівнів сервісу в складній інфраструктурі. Значне число новинок буде представлено програмними продуктами, які розширюють функціональність, спрощують адміністрування, підвищують продуктивність СЗД і рівень автоматизації процесів.

Збережеться й тенденція переносу функціональних можливостей рішень старшого класу (віртуалізації, динамічного розподілу ємності зберігання, «знімків» даних, розбивки на розділи, створення логічних томів або наборів даних, міграції даних між дисковими масивами, віддаленої реплікації для захисту від катастроф і ін.) у системи велике низької цінової категорії. Це виразиться в появі систем зберігання середнього цінового класу, що володіють надійністю, продуктивністю й функціональністю СЗД корпоративного рівня.

Перенос функціональних можливостей систем старшого класу в середній буде поширюватися й на системи початкового рівня. Тому деякі замовники, прагнучи скоротити витрати, віддадуть перевагу системам класом нижче.

У системах зберігання даних корпоративного класу ще більш актуальним напрямком стане застосування дедуплікації для різних систем вторинного зберігання – резервного копіювання, архівування й т.п. Дедуплікація даних на

джерелі й об'єкті – важливий метод скорочення операційних витрат на керування й захист даних, зниження вимог до ємності зберігання. Технологія дедуплікації буде застосовуватися в переважній великості процедур резервного копіювання. Відповідно до оцінок вендорів, технології віртуалізації, дедуплікації й керування життєвим циклом даних допомагають знизити ТСО на 30-50%.

У якості одного зі способів скорочення витрат на довгострокове зберігання даних і архівування як і раніше будуть розглядатися стрічкові бібліотеки. Щільність запису на магнітну стрічку продовжує рости. Ємність випущених недавно картриджів LTO 6 становить 6,25 Тбайт (при стиску даних). Для деяких категорій замовників оптимальним варіантом може стати зберігання даних у хмарі, але в Україні великість власників підприємств не готові передати контроль над IT-активами й корпоративним даними зовнішнім провайдерам послуг. Хмарні послуги усе ще перебувають у стадії розвитку, однак у найближчому майбутньому цей сегмент вплине на ринок СЗД. «Виробники стикнуться з великими труднощами, якщо через рік вони не зможуть дохідливо пояснити замовникам, навіщо їм треба купувати встаткування, коли можна з мінімальними витратами організувати зберігання даних у хмарі», – уважають в IDC.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення ієрархічних систем зберігання даних з використанням технології Tiered Storage, 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

IBM Storwize® V3700

IBM Storwize® V3700, створений на основі ПЗ IBM® Spectrum Virtualize™ – це дискова система зберігання початкового рівня зі складними функціями, які звичайно не підтримуються для систем такого класу. Вона допомагає організаціям підвищити ефективність і гнучкість за рахунок оперативного виділення ресурсів і перенесення даних з існуючих систем без порушення роботи. Завдяки інноваційним технологіям, що лежать в основі всіх продуктів сімейства Storwize, Storwize V3700 відповідає вимогам малих і середніх підприємств, які шукають блокове сховище за доступною ціною.

Переваги IBM Storwize V3700:

– Доведена простота керування, функціональність і сумісність Storwize.

– Простота використання: графічний інтерфейс користувача й функції керування системою за принципом "вибери й клацни" нового покоління забезпечують максимально просте керування даними.

– FlashCopy®: створення моментальних копій даних для резервного копіювання або тестування додатків.

– Оперативне виділення ресурсів:підтримка бізнес-додатків, яким потрібне значне розширення, при витраті тільки необхідного обсягу пам'яті.

– IBM Easy Tier®: забезпечує велике високу продуктивність із одночасним контролем витрат.

– Внутрішня віртуализація: допомагає швидко реагувати на мінливі вимоги.

– Міграція без порушення роботи:прискорює впровадження й мінімізує простої.

– Реплікація IP: знижує витрати на віддалений дзеркальний захист за рахунок інноваційних методів оптимізації мережі.

– Робота в турбо-режимі: підвищує максимальну кількість операцій уведення-виводу в секунду й продуктивність.

Особливості:

– Легкість керування й розгортання системи зберігання даних за допомогою сучасного графічного інтерфейсу користувача.

– Ефективність завдяки віртуализації внутрішніх ресурсів системи зберігання й високопродуктивної технології ощадливого виділення ресурсів (thin provisioning).

– Безперервний доступ до даних за рахунок інтегрованої функції міграції даних без переривання роботи системи.

– Поліпшення використання мережних ресурсів для віддаленого дзеркалювання завдяки інноваційній технології реплікації.

– Оптимізація витрат для змішаних робочих навантажень завдяки підвищенню продуктивності втроє з використанням усього 5% флеш-пам'яті в загальній ємності, а також технології IBM® Easy Tier1.

– Система відповідає стандартам Network Equipment Building System (NEBS) і European Telecommunications Standards Institute (ETSI).

– Подібний рівень надійності й розширених функцій звичайно властиві тільки системам велике високого класу.

– Можливість масштабування до 240 2, 5-дюймових дисків або до 120 3, 5-дюймових дисків при використанні 9 модулів розширення.

– Можливості підключення до хосту за допомогою портів serial attached SCSI (SAS) 6 Гбіт/с і Internet Small Computer System Interface (iSCSI) 1 Гбіт/с (у стандартній комплектації).

– Енергозберігаючі функції допомагають знизити енергоспоживання.

В епоху хмарних середовищ, Великих даних і аналітики, мобільних і соціальних обчислень організаціям необхідно задовольняти постійно, що змінюються вимоги, до зберігання даних, а також поліпшувати економічні показники їхнього використання. IT-середовище повинна швидше й ефективніше надавати велику кількість сервісів, підтримувати аналіз у реальному часі й розширювати можливості взаємодії із замовниками. Правильно сформована інфраструктура забезпечує можливість обміну інформацією, захисту транзакцій і аналізу в режимі реального часу.

Сімейство рішень IBM Storwize, створене з використанням ПЗ IBM Spectrum Virtualise, включаючи систему IBM Storwize V3700, допомагає організаціям поліпшити економічний ефект від використання даних завдяки підтримці нових робочих навантажень, важливих для успішної роботи. Сімейство систем Storwize дозволяє обробляти великі обсяги даних з мобільних і соціальних додатків, підтримує швидкий і гнучкий розподіл хмарних сервісів і забезпечує продуктивність і масштабованість, необхідні для нових аналітичних технологій.

IBM Storwize V3700, система початкового рівня в сімействі IBM Storwize, – це ефективна, легка в експлуатації система, розроблена для доповнення віртуальних серверних середовищ і забезпечення гнучкості й інноваційних функцій зберігання даних, надаваних ПЗ Spectrum Virtualise.

Характеристики продукту:

– Графічний користувальницький інтерфейс на основі веб-браузера з можливостями керування за допомогою миші.

– Функція віртуализації внутрішньої дискової системи зберігання уможливлює швидке й гнучке надання ресурсів і простої внесення змін у настроювання системи.

– Технологія ощадливого розподілу ресурсів дозволяє динамічно масштабувати додатка з використанням тільки фактично необхідного обсягу пам'яті.

– Простий процес міграції даних із зовнішнього пристрою зберігання в систему Storwize V3700 (однобічна міграція із зовнішнього пристрою).

– Технологія FlashCopy дозволяє створювати моментальні копії додатків з метою їхнього резервного копіювання або тестування.

– Віддалене дзеркалювання з використанням недорогого IP-з'єднання дозволяє ефективно створювати резервні копії даних для забезпечення їхньої схоронності.

– Допомога в забезпеченні відповідності специфікаціям NEBS і ETSI.

Короткий опис апаратних засобів:

– Модульна дискова система у форм-факторі 2U.

– Зовнішній інтерфейс SAS 6 Гбіт/с і iSCSI 1 Гбіт/с і додаткові хост-порти Fibre Channel (FC) 8 Гбіт/с, iSCSI/Fibre Channel over Ethernet (FCo) 10 Гбіт/с або додаткові порти SAS 6 Гбіт/с і iSCSI 1 Гбіт/с.

– Ємність до 960 терабайт (ТБ).

– Максимальна кількість жорстких дисків.

– Модуль дисків малого форма-фактора: 24 2, 5-дюймових диски.

– Модуль дисків великого форма-фактора: 12 3, 5-дюймових дисків.

– Підтримує до 9 модулів розширення (до 240 дисків на систему).

– Двопортові жорсткі диски 6 Гбіт/с із інтерфейсом SAS з можливістю «гарячої» заміни.

– Підтримка RAID рівнів 0, 1, 5, 6 і 10.

– Кеш-пам'ять 4 ГБ (на контролер) у стандартній версії, можлива модернізація до 8 ГБ.

– Резервовані джерела живлення й модулі охолодження з можливістю «гарячої» заміни.

– Джерело змінного струму (110-240 В).

– Джерело постійного струму (-48 – -60 В), доступний для модулів SFF.

Hitachi Command Suite Data Mobility

Hitachi Command Suite Data Mobility має повний набір функцій для переміщення даних, сполучаючи в собі можливості рішень Hitachi Dynamic Tiering і Hitachi Tiered Storage Manager. Таке сполучення дозволяє виконувати інтелектуальне розміщення даних, міграцію томів без переривання роботи й оптимізацію рівнів обслуговування у віртуалізованих середовищах зберігання Hitach.

Програмне забезпечення Dynamic Tiering спрощує адміністрування системи зберігання, усуває складності, пов'язані з керуванням життєвим циклом даних, і оптимізує використання багаторівневої системи зберігання.

Підвищення продуктивності й скорочення витрат завдяки автоматичному розміщенню даних.

Автоматична оптимізація продуктивності й використання простору.

Автоматична оптимізація продуктивності за рахунок рівномірного розподілу даних по пулах ресурсів.

За допомогою Tiered Storage Manager можна завчасно зіставляти вимоги з погляду вартості, продуктивності й доступності бізнес-додатків з характеристиками ресурсів зберігання даних.

Простий вибір оптимальних рівнів зберігання з урахуванням потреб у використанні даних і продуктивності.

Відсутність необхідності в аналітичних навичках.

Керування всіма типами даних, платформами зберігання Hitachi і сумісними віртуалізованими системами зберігання сторонніх виробників.

Функціональні можливості:

– Підвищення продуктивності.

– Забезпечення продуктивності на рівні, близькому до Tier 0, з одночасним скороченням витрат завдяки сполученню сховищ на флеш-накопичувачах і SAS.

– Автоматичне переміщення найбільш активних даних на рівень із максимальною продуктивністю.

– Активна флеш-пам'ять, що забезпечує переміщення сторінок з високою деталізацією на рівні з велике високою продуктивністю за секунди або навіть частку секунд.

– Підвищення ефективності й пропускної здатності системи зберігання за допомогою гранулярного переміщення блоків (сторінок) даних.

– Рівномірний розподіл даних по всіх пулах ресурсів зберігання.

– Автоматична оптимізація розміщення даних з використанням коефіцієнта активності на основі числа операцій уведення-виводу для підвищення продуктивності.

– Скорочення витрат на зберігання.

– Скорочення витрат на носії за рахунок автоматичної оптимізації використання рівнів сховища.

– Ефективне використання простору завдяки «тонкому» виділенню ресурсів.

– Відсутність необхідності класифікувати дані вручну.

– Зниження вимог до площі приміщень, енергозабезпеченню й охолодженню.

– Підвищення ефективності адміністрування

– Автоматичне переміщення необхідних даних у потрібне місце в необхідний момент часу.

– Коректування з урахуванням динамічного навантаження й вимог до ємності.

– Переміщення сторінок нагору й долілиць для динамічної оптимізації розміщення з урахуванням їх коефіцієнтів активності.

– Рівні зберігання, що набудовуються користувачем, з урахуванням потреб додатків.

– Міграція томів між рівнями зберігання без переривання роботи.

– Профілі розміщення даних, у яких користувальницькі параметри зберігаються як шаблони для багаторазового застосування.

Переваги:

– Підвищення адаптуємості навантажень шляхом швидкого переміщення даних, які раптово стали активними, на відповідні шари флеш-памяти.

– Оптимізація розміщення даних відповідно до вимог до рівнів обслуговування додатків.

– Скорочення витрат за рахунок переносу рідко використовуваних даних на економічні рівні зберігання.

– Оперативне виділення ресурсів зберігання для нових додатків.

– Оптимізація відновлення технологій і систем завдяки міграції даних без переривання роботи.

– Істотна економія часу адміністрування й поліпшення результатів.

Hitachi Tiered Storage Manager

Продукт

Hitachi Tiered Storage Manager для мейнфреймів – нове керуюче ПЗ, розроблене для IBM z/OS і які реалізує технологію динамічного переміщення даних по рівнях зберігання Hitachi Dynamic Tiering у середовищі мейнфреймів.

Нове ПЗ управляє рівнями обслуговування сховищ даних для найважливіших додатків, установлених на мейнфреймах, і за допомогою технології Hitachi Dynamic Tiering автоматично оптимізує продуктивність і завантаження багаторівневого сховища. Завдяки онлайновому керуванню рівнями обслуговування замовники можуть підвищити продуктивність і ефективність функціонування всього середовища, а також запобігати виникнення проблем.

Програмне забезпечення Hitachi Tiered Storage Manager для мейнфреймів інтегровано зі знайомими адміністраторам інструментами DFSMS, що дозволяє обійтися без додаткових інтерфейсів і звести до мінімуму навчання. Для автоматизації керування рівнями обслуговування нове ПЗ використовує політики

автоматичного переміщення даних по рівнях зберігання, певні для наборів даних додатків відповідно до наявних груп систем SMS або окремими пристроями. У той же час воно надає функції створення звітів і контролю, забезпечуючи керування й безперервний моніторинг функціонування технології Hitachi Dynamic Tiering у середовищі мейнфреймів.

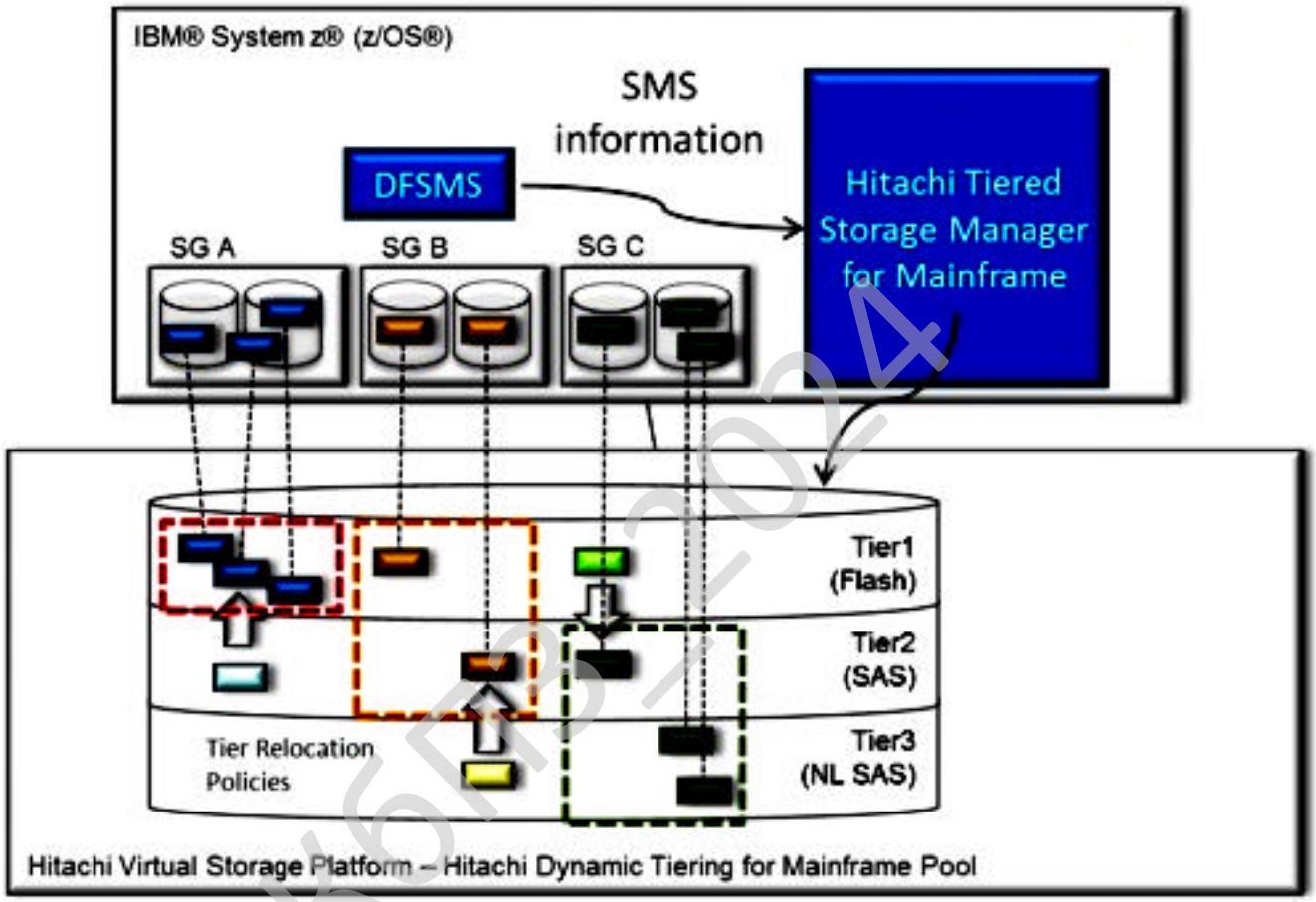

Рисунок 2.1 – Структура Hitachi Tiered Storage Manager

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Embarcadero Delphi, раніше Borland Delphi і Codegear Delphi, – інтегроване середовище розробки ПЗ для Microsoft Windows, Mac OS, iOS і Android мовою Delphi (що раніше носила назву Object Pascal), створена спочатку фірмою Borland

і на даний момент приналежна й розроблювальна Embarcadero Technologies. Embarcadero Delphi є частиною пакета Embarcadero RAD Studio і поставляється в чотирьох редакціях: Community (поширюється безкоштовно й має обмежену ліцензію на використання в комерційних цілях), Professional, Enterprise і Architect.

Delphi 10.4 Sydney

Випущено 26 травня 2020 року. RAD Studio Delphi 10.4 забезпечує значно поліпшену високопродуктивну нативну підтримку Windows, кращу продуктивність розробки, миттєві підказки code completion, прискорення виконання коду із синтаксисом керованих записів, поліпшення виконання паралельних завдань на сучасних багатоядерних CPU, а також містить більш 1000 виправлень багів, поліпшення продуктивності середовища й бібліотек і багато чого крім того.

Основні можливості Delphi 10.4.1:

− Істотні розширення для Windows: поліпшення для застосунків на моніторах 4K High DPI, інтеграція з новим WebView2 на базі Chromium, використання розширених title bars, таких же, як в Office, Explorer, Google Chrome.

− Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

− Істотне поліпшення Delphi Code Insight (без можливого блокування IDE – в окремому процесі), що допоможе при роботі з великими проектами.

− Тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання.

− Розширена підтримка бібліотек C++: ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode.

− Відладник Win 64 (на LLDB) і збирач для C++.

− Поліпшення для C++: Включена велика кількість поліпшень STL з Dinkumware.

− Підтримка Metal Driver GPU для macOS і iOS.

− Вбудований Fmxlinux.

− Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API. Реалізація компонента Media Player для macOS тепер використовує Avfoundation. Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

− Численні поліпшення швидкості й стабільності роботи нашої бібліотеки The Parallel Programming Library (PPL).

− Додані оновлені драйвери для FireBird, PostgreSQL і SQLite.

− Клієнтські бібліотеки HTTP і REST Client розширені застосунковими можливостями роботи з HTTPS. Також були розширені можливості підтримки Amazon AWS services

− У технологію Visual LiveBindings внесена безліч поліпшень, у тому числі швидкодії, що стосуться, застосунків на VCL і FireMonkey

RAD Studio 10.4 Короткий огляд:

– Істотні розширення для Windows. Створення застосунків, що чудово виглядають, із чіткими елементами інтерфейсу на 4k моніторах High DPI за допомогою нової гнучкої підтримки стилів елементів керування на екрані. Інтеграція із сучасними, безпечними web-технологіями від Microsoft – новим WebView2 на базі Chromium. Використання сучасних розширених title bars, таких же, як в Office, Explorer, Google Chrome, у своїх проектах. Істотні поліпшення надійності налагодження в новому відладнику для C++ Windows 64-bit.

– Зросла продуктивність розробки. Ріст продуктивності за рахунок миттєвої реакції підказок code completion у середовищі IDE. Краща сумісність із уже наявною кодовою базою, і спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю. Швидке зв'язування даних і візуальних елементів за допомогою розширеної технології Visual LiveBindings з підвищеною швидкодією. Просте використання розповсюджених бібліотек C++,

наприклад, ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode. Оновлена підтримка Amazon AWS cloud.

– Поліпшення швидкодії і якості. Більш 1000 поліпшень швидкодії і якості. Краща ефективність коду за допомогою нового синтаксису custom managed records. Більш швидке виконання паралельних завдань на сучасних багатоядерних CPU. Переконаєтеся в прискоренні відображення на екрані з підтримкою Metal API на macOS і iOS. Краща сумісність із уже наявною кодовою базою й спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю.

Істотне поліпшення Delphi Code Insight

Як найбільше й головне поліпшення інструментів програмування Delphi за багато років, в 10.4 Delphi Code Insight реалізований через Language Server Protocol (LSP). LSP – це технологія генерації результатів для code completion, навігації й інших сервісів в окремому процесі. Це значить, що code completion і Code Insight одержать більш точні результати без блокування IDE. 10.4 забезпечує набагато більш високу продуктивність розроблювачів, які працюють із більшими проектами, що містять мільйони рядків коду.

Delphi Custom Managed Records

Ключове розширення мови Delphi: тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання. Управляйте тем, як ці структури створюються, копіюються й звільняються з допомогу вашого коду, який буде виконуватися у відповідний момент.

Це розширює потужність конструкцій records в Delphi, які використовуться щоб одержати більшу ефективність у порівнянні із класами.

Єдине керування пам'яттю

Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

У порівнянні з Automatic Reference Counting (ARC), це дає кращу сумісність із існучим кодом і спрощує написання компонентів, бібліотек і застосунків.

ARC модель керування пам'яттю model залишилася для керування рядками й посиланнями на тип інтерфейсу на всіх платформах. Для C++ це означає, що при створенні й звільненні Delphi-style класів в C++ використовується звичайне керування пам'яттю, як у будь-якого heap-allocated класу C++, що значно знижує складність коду.

Розширена підтримка бібліотек C++

В 10.4 ми портували багато популярних бібліотек C++ у C++Builder.

Забезпечивши оптимізовану підтримку бібліотек ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode, поряд із уже підтримуваними Boost і Eigen, які можуть бути додані за допомогою менеджера пакетів Getit.

Win 64-відладник і збирач для C++

В 10.4 з'явився новий відладник C++ для Windows 64-bit. Відладник заснований на LLDB і показує значне збільшення стабільності при налагодженні 64-bit застосунків поряд з новими відладочними можливостями, такими як перегляд і інспекція типів начебто рядків C++ і Delphi, а також колекцій STL, включаючи std::vector, std::map і інших. Крім того, згенерована для застосунку відладочна інформація має інший внутрішній формат, сприяючи більш стабільному й багатому на можливості процесу налагодження, більш докладним перегляду й інспекції в debug-time.

Підвищення якості й швидкодії інструментів

- Велика кількість поліпшень STL від Dinkumware.
- Поліпшені деякі найважливіші методи й області RTL, на базі поліпшень сумісності з популярними бібліотеками C++.
- Поліпшена підтримка Cmake.
- Велика кількість виправлень для підвищення стабільності і якості.

– Відновлення Windows API – Обновлено й додали безліч декларацій API щоб добитися ще більшої інтеграції із платформою Windows.

– Загальні вдосконалення в бібліотеці доступу до БД FireDAC, включаючи оновлені драйвера для FireBird, PostgreSQL і SQLite. Вибір статичного або динамічного підключення SQLite до застосунку.

Змінені стилі VCL для High DPI

В 10.4, архітектура стилізації VCL була суттєво розширена для підтримки High DPI і 4K моніторів. Тепер усі елементи UI на формі VCL автоматично масштабуться під відповідне до монітора дозвіл для показу форми. Був оновлений API стилізації для підтримки стилів high DPI.

Кожний графічний елемент UI може бути обраний з наборів різних масштабів і масштабований до потрібного DPI, що дає чітке зображення елементів UI на всіх моніторах.

Нові High DPI стилі й стилізація окремих VCL компонент

Обновлено велике число вбудованих і преміальних VCL стилів для підтримки нового режиму стилізації High-dpi. Це дозволяє вам створювати застосунку з відмінним дизайном для всіх моніторів.

Розроблювачі VCL застосунків тепер можуть використовувати трохи VCL стилів на різних формах в одному застосунку або в різних компонентів на одній формі. Це також включає стилізацію компонентів загальною темою для платформи. Крім застосункової гнучкості використання стилів, це дозволяє використовувати нестилізуємі компоненти із зовнішніх бібліотек в VCL застосунках, що використовуть стиль.

Поліпшена кроссплатформеність

– Додана підтримка Metal Driver GPU для macOS і iOS.

– Крім підтримки останнього iOS SDK, в RAD Studio 10.4 розроблювачі можуть задовольнити нові вимоги Apple до набору стартових екранів.

– Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

– Користувачам редакцій Enterprise або Architect доступна повна інтеграція Fmxlinux з IDE для створення клієнтських застосунків Linux з GUI.

– Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API.

– Реалізація компонента Media Player для macOS тепер використовує Avfoundation.

Оновлений менеджер пакетів Getit

Менеджер пакетів Getit в IDE був значно вдосконалений.

Дати випуску релізів пакетів тепер видні, і можливе сортування списку по цих датах; відбір тільки встановлених пакетів, контенту, доступного тільки при наявності підписки, багато чого іншого.

Універсальний інсталятор для установки Online і Offline

В 10.4 включений новий універсальний інсталятор, який використовує технологію на базі Getit. Цей інсталятор підтримує як online, так і offline (з ISO) варіанти установки.

Тепер обоє варіанта установки дозволяють вам указати початковий набір можливостей RAD Studio для установки, наприклад, свою комбінацію мов програмування й цільових платформ, мов інтерфейсу, і додавати до нього або видаляти непотрібне в будь-який момент.

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 ієрархічних систем зберігання даних з використанням технології Tiered Storage.

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

Сьогодні індустрія систем зберігання даних (СЗД) стоїть перед викликом: обсяги комерційної інформації продовжують рости, але організації змушені заощаджувати на всьому, скорочуючи бюджети й персонал. Перед IT-службами ставлять завдання оптимізувати інфраструктуру зберігання, іншими словами – знизити витрати на її підтримку без погіршення якості сервісу. Неоднорідність і нерівноцінність даних уже стає очевидним і основним ресурсом оптимізації – через структурування інформації й обслуговуючих пристроїв.

На сортуванні даних по цінності, доступності, затребуваності побудовані такі популярні концепції, як багаторівневе зберігання (tiered storage), ієрархічне керування системами зберігання (HSM), керування життєвим циклом інформації (ILM). Техніки переміщення даних з дорогих носіїв на великі дешеві по встановлених принципах і сценарію є суттю керуючого ПЗ. Воно ж вносить основний вклад у вартість високорозвинених СЗД. Великі гроші беруть не за встаткування, а за адресне життєзабезпечення активних даних і здатність вчасно розвантажувати СЗД верхнього ешелону від застаріваючої й інформації, що знецінюється.

Життєвий цикл інформації

Будь-яка інформація, незалежно від формату й додатків, що породжують, проходить на своєму життєвому шляху через чотири стани:

– Динамічні дані оперативного доступу. Це об'єкт і продукт роботи активних додатків, для них характерні постійні зміни. Перебуваючи у фокусі множинних конкурентних запитів, вони вимагають максимальних зусиль по забезпеченню продуктивності, доступності, безпечного зберігання.

– Статичні дані оперативного доступу. Зміни в них уже не вносяться, але усе ще потрібно їхня висока готовність. Відсутність виправлень – сигнал до переміщення цієї інформації на пристрої, менш дорогі в обслуговуванні. Послабляючи вимогу негайного доступу, ми одержуємо ще велику волю вибору підтримуючої СЗД.

– Статичні дані відкладеного доступу. Такі дані не тільки не змінюються, але й рідко запитуються, однак по необхідності повинні бути доставлені додатку за розумно короткий час. У подібному стані перебуває велика частина скільки-небудь значимої інформації, тут ціна реалізації зберігання виявляється ключовим фактором вибору підтримуючої СЗД.

– Резервний архів. Інформація, що зберігається на випадок аварійного відновлення. При гіршому зі сценаріїв повинна існувати повна копія даних неважливо на якому з носіїв: дисках, стрічці або оптиці.

На кожній з перерахованих стадій пред'являються свої вимоги до продуктивності й часу відгуку обслуговуючих СЗД, розраховуються свої оцінки ризику втрат і простою, працює своя економіка. Керування еволюцією даних полягає в тому, щоб призначити кожній одиниці інформації місце зберігання, що відповідає її цінності, у кожний момент її життєвого циклу. Ідея переносу застаріваючих даних на менш дорогі пристрої прямо треба з якісного аналізу даних. Здешевлення зберігання з урахуванням особливостей цільових ринків, витрат на організацію планової міграції, що міняються ризиків – основний виклик індустрії зберігання даних.

Багаторівневе зберігання (tiered storage)

Багаторівневе зберігання (tiered storage) – спосіб керування інформацією, що розростається постійно, приведення інфраструктури у відповідність із якісними й об'ємними характеристиками даних, що зберігаються. Ієрархічне розшарування СЗД по рівнях Tier 1 – Tier 2 (– Tier 3) виходить із вимог бізнесу до продуктивності, безперервності, безпеці, захисту, схоронності, чутливості до ціни реалізації. Метою є вивільнення дорогих ресурсів Tier 1 і зниження витрат на

зберігання за рахунок переміщення статичних даних на пристрої нижніх рівнів Tier 2 (Tier 3) – наприклад, дискові масиви на велике дешевих високоємних накопичувачах SATA.

У дійсності не так просто домогтися економії одним лише сортуванням даних і наступною міграцією на відповідний їх статусу пристрій. По-перше, немає галузевих стандартів. Зрозуміло, всі провідні вендори СЗД мають у своєму арсеналі розвинені інструменти міграції інформації між пристроями різних рівнів. Але працюють вони найчастіше тільки із платформами конкретної компанії. Тому вимога однорідності встаткування по всій вертикалі СЗД прив'язує користувача до постачальника й у якімсь ступені робить його заручником. По-друге, недосконалі апаратні технології відділення «гарячих» даних від «холодних», що приводить до витрати дорогого ресурсу високопродуктивних носіїв на малоцінну інформацію. По-третє, процес регулярної міграції даних відволікає обчислювальні ресурси серверів і СЗД, а масова переадресація шляхів звертання до даних у складних додатках залишає широке поле для помилок і збільшує ризик втрати інформації.

Реальний виграш може дати проектування ефективної інфраструктури СЗД зворотними методами: від специфіки завдання, а не можливостей певної лінійки СЗД; на основі відкритих систем замість закритих; уникаючи надмірності й надмірностей – на користь стандартних засобів керування. Природно, у критичні для бізнесу додатках повинні використовуватися адекватні за вартістю й функціональністю рішення Tier 1 від лідерів індустрії. У цьому найбільш дорогому сегменті ринку не вщухають «священні війни» брендів. Однак споровши, де зберігати статичну й застаріваючу інформацію, і там давно немає. Очевидно, що ресурси масивів фронтального ряду Tier 1 застосовувати для цього марнотратно. Зустрічаються різні підходи до організації Tier 2 (Tier 3), але в цілому можна сказати, що завдання зберігання неактивних даних обійдена увагою. Обслуговуючі їх СЗД називають не інакше як «допоміжними», «другорядними», «другим ярусом зберігання». Проте будь вони хоч «кухнею»

зберігання, хоч «прикомірком», це не применшує їхньої ролі й цінності для бізнесу.

Породжувати великі потоки даних можуть додатки й регламент ведення бізнесу. Так, істотне навантаження на СЗД дає документообіг. Тривале консервування може знадобитися вже не мега- або гіга-, а терабайтам оцифрованих документів, особливо якщо їх треба зберігати роками, що насамперед як діє посилальний ресурс. Після фази активного використання величезні обсяги даних осідають в архівних сховищах банків, страховиків, юридичних центрів, кадастрових бюро, логістичних компаній, медичних установ.

Конвергенція традиційних систем безпеки (відеоспостереження, контроль доступу, охорона периметра) з IT-інструментами (розпізнавання образів, системи ухвалення рішення) тільки збільшила вимоги до СЗД. Обсяги споживання дискового простору тут ростуть пропорційно запитам до дозволу зображення, кольоровості, частоті відновлення кадрів, тривалості строку зберігання архівів відеоспостереження. Подібним образом використовуються СЗД у центрах обробки викликів компаній і службах телефонної підтримки клієнтів, де запис переговорів операторів із клієнтами – обов'язкова складова частина внутрішнього регламенту.

Мультимедійний контент взагалі є одним з основних двигунів ринку СЗД високої ємності. Взяти відеовиробництво. Перше ніж потрапити на очі глядачів, «кілометри цифрової кіноплівки» знімаються, записуються, обробляються й звертаються усередині студій, породжуючи на кожному етапі розгалуження й резервні копії. Кілька днів роботи «у поле» забиває даними масив у десятки терабайт. Перехід медіаіндустрії на формати високої якості (HD-відео й багатоканальний звук) першою справою прискорює «проїдання» дискового простору. Звичайно, горезвісна безперервність бізнесу (і відповідні СЗД як центральний елемент інфраструктури) для виробничих студій важлива. Але куди велике гострою проблемою для них є забезпечення колективного доступу до великих обсягів даних при розумному ступені їхньої захищеності, з розумною

ціною реалізації. Того ж хочуть масштабні споживачі «відеоконсервів», ті ж файлообміні мережі.

Помітний внесок у розширення попиту на СЗД внесли основні драйвери сучасного корпоративного ринку – консолідація й віртуализація IT-інфраструктури. Концентрація даних і додатків природно веде до збільшення обсягів централізованого зберігання, підвищуючи при цьому вимоги до керованості, які ростуть із велике активним застосуванням у бізнесі засобів групового редагування документів (з підтримкою версій) і засобів мультимедіа (презентацій, ділової графіки, відеороликів, підкастів). Усе ширше застосовуються багатомірні аналітичні бази даних. Одержали розвиток класичні завдання, покладені на файлові (NAS) сервери: зберігання профілів користувачів, їх локальних поштових скриньок, документів. Автоматичне збереження попередніх копій документів сучасними ОС, пересилання в поштових вкладеннях маси файлів, зберігання образів віртуальних машин – все це теж збільшує вимоги до ємності СЗД.

Дисковий масив як будь-яке сховище інформації вимагає спеціалізованих програмних і апаратних засобів. У масивах Tier 2 немає повні апаратні дублювання всіх компонентів. Надійне зберігання даних на них забезпечує широкий набір службових процедур. Найпоширеніша й часто використовувана функція захисту від втрати інформації (якщо більш точно, від випадкового псування останніх екземплярів даних) – Snapshot, тобто фактично створення «миттєвих знімків» стану тому на певний момент, після чого всі зміни записуються як доповнення щодо цієї крапки. До неї користувач завжди може відкотитися. Сучасні масиви зберігають десятки знімків (Snapshot), що дає досить докладну хронологію змін.

Статичність даних до крапки фіксації знімка дозволяє легко реалізувати їхнє резервне копіювання на інший фізичний масив або том. Це можна робити не перериваючи роботи й не зупиняючи доступ до того за допомогою функції «тіньового» (не відчутного для користувачів) створення повної копії даних –

ShadowCopy. Можливості Snapshot і ShadowCopy поєднує функція Snapclone – формування миттєвих фізичних копій розділів. Велике універсальна функція Data Replication створює образи вихідних даних на інших томах або пристроях, хоча при цьому знижує продуктивність СЗД і припиняє доступ до інформації. Функція Mirrorclone виконує безперервне копіювання розділів у процесі роботи. Для вимогливих до надійності користувачів у деяких ОС для дискових масивів передбачена програмна функція кластеризації – Clustering. Вона застосовується для забезпечення безпеки зберігання даних і для балансування навантаження між вузлами кластера. Технологія Single-Instance Storage (SIS) виключає резервне копіювання дублюючої інформації на загальних томах – заощаджує місце на дисках.

При додаванні дисків або дискових полиць застосовуються технології Online RAID capacity expansion and RAID level migration (розширення RAID додаванням нових дисків для збільшення ємності масиву й одночасної зміни рівня RAID), Drive insertion/removal detection and rebuilding (автоматичне визначення видалення й вставки диска, запуск процедури відновлення) і Background initialization (негайна доступність дисків і їхня фонова ініціалізація). Змінити рівень RAID і його властивості (розміру смуги даних stripe size, алгоритму роботи з кеш-пам'яттю) можна за допомогою Online RAID level/stripe size migration. За раціональним застосуванням користувачами простору доглядають служби Quota management (квотування, обмеження на розмір області даних, що охороняє від вичерпання дискового простору) і Content filtering (фільтрація змісту, розмежування доступу користувачів по типах файлів, що дозволяє адміністраторам обмежувати види даних, до яких забезпечується спільний доступ).

Описані функції не є обов'язковим атрибутом всіх дискових масивів, а їхня суть і розмаїтість не пов'язані з обсягом зберігання, приналежністю до певного шару зберігання або бренду. Це розмаїтість – робочий інструментарій адміністратора й зайве свідчення тому, що надійне зберігання не зводиться до

одного тільки забезпечення апаратної відтказостійкості й боротьбі за мінімум крапок відмови. Доречним рішенням і способом здешевлення тут може бути використання відкритих стандартів, замінних блоків і компонентів рішень, можливо, велике стандартних програмних засобів.

Визначити оптимальний сценарій зберігання даних в ієрархічній структурі дозволить простий алгоритм у кілька кроків, які бажано виконати до проектування нових СЗД:

1. Класифікувати дані для багаторівневого зберігання з урахуванням вимог бізнесу.

2. Побудувати активний архів. Визначити правила переміщення в нього рідко використовуваних або статичних даних.

3. Позбутися від повторюваних даних, що прямо вплине на ціну зберігання.

4. Настроїти процедури резервного копіювання й відновлення даних.

5. Використовувати Snapshot-повні копії робітників даних. Це допоможе резервуванню й відновленню інформації (хоча й додасть навантаження на диски й позначиться на ціні).

6. Застосувати віртуализацію. Віртуализація додатків і систем зберігання значно поліпшує показники утилізації потужностей серверів і СЗД і, в остаточному підсумку, заощаджує засобу.

Сервери DSS у структурі зберігання

Традиційно компанії вишиковують шари зберігання послідовно: від старших Tier 1 до молодших Tier 2 (Tier 3). Зробивши інвестиції в СЗД Tier 1, багато хто продовжують дооснащувати їхніми зовнішніми полками розширення, у тому числі на недорогих дисках SATA. Але масиви Tier 1 нерівноцінні й теж діляться по класах «entry level – midrange – enterprise». СЗД початкового рівня не мають запасу продуктивності керуючих модулів і розширюваності (кількості зовнішніх полиць, що підключаються,). У цьому випадку й великий обсяг дискового простору під другорядні додатки не додати, і ресурс відбирається в

основних, критичних додатків. Купувати ще один масив початкового рівня Tier 1? Переходити на midrange? Кожної зі сценаріїв зажадає чималих додаткових вкладень. Можливо, проблема надійного зберігання великого обсягу даних формулювалася невірно – у рамках стратегії прямолінійного розвитку інфраструктури або в силу прихильності бренду.

Загалом кажучи, на ринку давно й широко представлений універсальний інструмент зберігання й гнучкого керування великими обсягами даних у складі корпоративної інфраструктури – сервери зберігання даних (DSS, Data Storage Server). Термін DSS увели виробники спеціалізованих ОС. Так, Microsoft почала похід на ринок пристроїв зберігання даних з ОС Windows Storage Server 2003 R2, що працює на файловому рівні. Її розвиток – платформа Windows Unified Data Storage Server 2003 – уже дозволяла обслуговувати мережні пристрої з файловим і блоковим доступом до даних. Німецький розроблювач програмного забезпечення для систем зберігання даних Open-E) багато років просував Open-E DSS, що поєднує функціональність NAS і iSCSI (притім що продавав і окремі інструменти Open-E NAS і Open-E iSCSI). Сьогодні Microsoft поширює через виробників OEM-серверів єдину ОС Windows Storage Server 2008, базові можливості якої поєднують функції файлового сервера з обслуговуванням процедур і блокового доступу до даних по протоколі iSCSI. Так само як і Open-E звела всі свої продукти в один універсальний DSS V6).

Сучасні програмні засоби керування зберіганням великих обсягів даних у мережі не зводяться до функціональності NAS-серверів. Це легко бачити по відношенню самих розроблювачів ПЗ до передбачуваної ролі продуктів. Колись Microsoft представляла Windows Storage Server 2003 R2 як «виділений сервер файлів і печатки на основі операційної системи Windows Server 2003 R2 і апаратний пристрої зберігання даних у мережі (NAS), призначений для побудови високонадійних, легко інтегрувальних і економічних мережних СЗД». Сьогодні компанія позиціонує Windows Storage Server 2008 як «комплексне рішення для

зберігання даних, що спрощує керування зберіганням інформації й знижує його вартість».

Фактично сервери DSS – це клас пристроїв на стику серверів x86 загального призначення й спеціалізованих СЗД. Як і всі сервери x86, вони побудовані по відкритих індустріальних стандартах. Схожими на СЗД їх робить велика кількість дискових накопичувачів, як наслідок – висока досяжна ємність на недорогих дисках SATA і програмні засоби керування зберіганням. Службові процедури – із числа описаних у попередній главі – реалізуються постачальником ОС або сторонніми виробниками сервісних утиліт. Таким чином, сервери DSS вирішують завдання зберігання великого обсягу даних, забезпечуючи при цьому:

– файловий (NAS) і блоковий (iSCSI, FC) доступ до даних;

– підтримку гетерогенної інфраструктури (клієнтів різних ОС);

– використання стандартних дисків SSD/SAS/SATA у різних комбінаціях;

– можливість дооснащення зовнішніми інтерфейсами (1Gb або 10Gb Ethernet, Fiber Channel, Infiniband) і їхніми комбінаціями;

– гарне масштабування за числом мережних підключень і ємності зберігання додатковими полками розширення.

В остаточному підсумку метою є створення ефективної інфраструктури зберігання. Ефективної – значить економічної, розширюваної, з еластичністю по числу підключень і прогнозованих сценаріїв росту. Ніякий бренд, навіть виконаний кращих намірів, не замінить самостійного й усвідомленого проектування середовища зберігання компанією-власником даних. Тільки облік всіх специфічних для бізнесу факторів (виділюваного бюджету; цінності інформаційних активів; вартості простою, підтримки; кваліфікації персоналу; адаптуємості до мінливих умов) змусить дані працювати на компанію, а не навпаки.

3.2 Розробка структурної схеми

Створення ієрархічних систем зберігання (Tiered Storage), що дозволяють автоматично переміщати дані між різними рівнями зберігання – накопичувачами, які відрізняються швидкодією й вартістю (SSD, диски SAS або SATA), допомагає оптимізувати вартість зберігання й зробити більш ефективним керування життєвим циклом інформації. Цей підхід, уже знайомий замовникам по таких технологіях, як EMC FAST і Dell Fluid Data, використовують все велике число вендорів. Всі частіше в якості одного з рівнів зберігання буде вибиратися хмара – такі рішення допомагають забезпечувати катастрофостійкість і архівування.

Динамічне багаторівневе зберігання дозволяє збільшити продуктивність додатків, скоротити витрати на зберігання даних і енергоспоживання. Такі технології, як дедуплікація даних, динамічний розподіл ємності, застосування накопичувачів SSD і автоматизоване багаторівневе зберігання даних, значно підвищують ефективність використання наявних ресурсів зберігання й знижують потреба в додатковій ємності. Сполучення технологій здатне дати кумулятивний ефект.

У міру здешевлення SSD і флешпам’яти, підвищення надійності й розвитку інших супутніх технологій інтерес до подібним до рішень стане зростати. Накопичувачі SSD корпоративного класу застосовуються в усі велике широкому спектрі СЗД. Застосування SSD дозволяє зробити СЗД компактніше й підвищити якість обслуговування (IOPS).

Продуктивність серверів росте істотно швидше, ніж швидкість виконання дискових операцій читання/запису, тому при проектуванні СЗД особлива увага приділяється продуктивності дискової підсистеми.

Поширення твердотільних накопичувачів може внести значні корективи: SSD будуть застосовуватися у великій кількості моделей серверів. Поряд з гібридними дисковими масивами (SSD+HDD) на ринку з'являться нові моделі масивів, де використовуються винятково флеш-накопичувачі.

Можна чекати подальшого росту попиту на багатопротокольні, уніфіковані СЗД. Їхня основна перевага полягає в одночасній підтримці декількох мережних протоколів, а також у можливості роботи не тільки з файлами по мережі Ethernet, але й із блоками даних по мережі Fibre Channel. Такі системи є в багатьох ведучих вендорів. Уніфіковані горизонтально масштабовані мережні системи зберігання даних позиціонуються як оптимальні рішення для віртуалізованої серверної інфраструктури. Вони дозволяють із однієї консолі управляти зберіганням образів ВМ незалежно від того, який протокол використовується. Впровадження уніфікованих СЗД спрощує міграцію даних і забезпечує ефективне спільне використання системи зберігання ресурсами й користувачами.

Внаслідок консолідації додатків і впровадження віртуалізації зростають вимоги до продуктивності і ємності СЗД.

Система зберігання повинна мати гарну горизонтальну масштабованість.

Тому попитом будуть користуватися системи, що передбачають можливість масштабування й нарощування як ємності, так і продуктивності, що не тільки забезпечить безперервну роботу додатків, але й запобіжить зниженню швидкості їхньої роботи з мері росту обсягів даних. Крім того, в IDC прогнозують, що середньорічний ріст ринку інфраструктурних технологій для Великих Даних складе 44%, і це теж вплине на розробку нових архітектур СЗД.

Аналітики відзначають усе велике активне просування інтегрованих рішень, у яких поєднуються програмна й апаратна частини, при цьому віртуалізація серверів не тільки створює нові можливості, але й ставить перед розроблювачами нові завдання. Так, наприклад, реалізовані дисковими масивами «знімки» даних будуть задіяні в багатьох програмних продуктах резервного копіювання/відновлення. Консолідація, віртуалізація критичних для бізнесу додатків, інтеграція засобів керування інфраструктурою із продуктами віртуалізації, виділення віртуальних ресурсів, підвищення енергоефективності, використання дедуплікації, керування життєвим циклом даних, а також

застосування флешнакопичувачів ставляться до числа основних технологічних напрямків розвитку СЗД.

Для реалізації програмного забезпечення зберігання даних з використанням технології Tiered Storage з підвищеною надійністю пропонується використовувати алгоритм LZW-стиску та перешкодостійкий алгоритм Хеммінга.

На рисунку 3.1 зображена структурна схема розробленого програмного забезпечення.

На ній відображено процес архівування та розархівування інформації за алгоритмом LZW, та підданя заархівованої інформації перешкодостійкому кодуванню за алгоритмом Хеммінга.

Якщо коротко, то алгоритм LZW діє наступним чином.

Даний алгоритм при стиску (кодуванні) динамічно створює таблицю перетворення рядків: певним послідовностям символів (словам) ставляться у відповідність групи біт фіксованої довжини (звичайно 12-бітні). Таблиця ініціалізується всіма 1-символьними рядками (у випадку 8-бітних символів – це 256 записів). У міру кодування, алгоритм переглядає текст символ за символом, і зберігає кожну нову, унікальну 2-символьний рядок у таблицю у вигляді пари код/символ, де код посилається на відповідний перший символ. Після того як нова 2-символьний рядок збережений у таблиці, на вихід передається код першого символу. Коли на вході читається черговий символ, для нього по таблиці перебуває вже, що зустрічався рядок, максимальної довжини, після чого в таблиці зберігається код цього рядка з наступним символом на вході; на вихід видається код цього рядка, а наступний символ використовується в якості початку наступного рядка.

Алгоритму декодування на вході потрібно тільки закодований текст, оскільки він може відтворити відповідну таблицю перетворення безпосередньо по закодованому тексту.

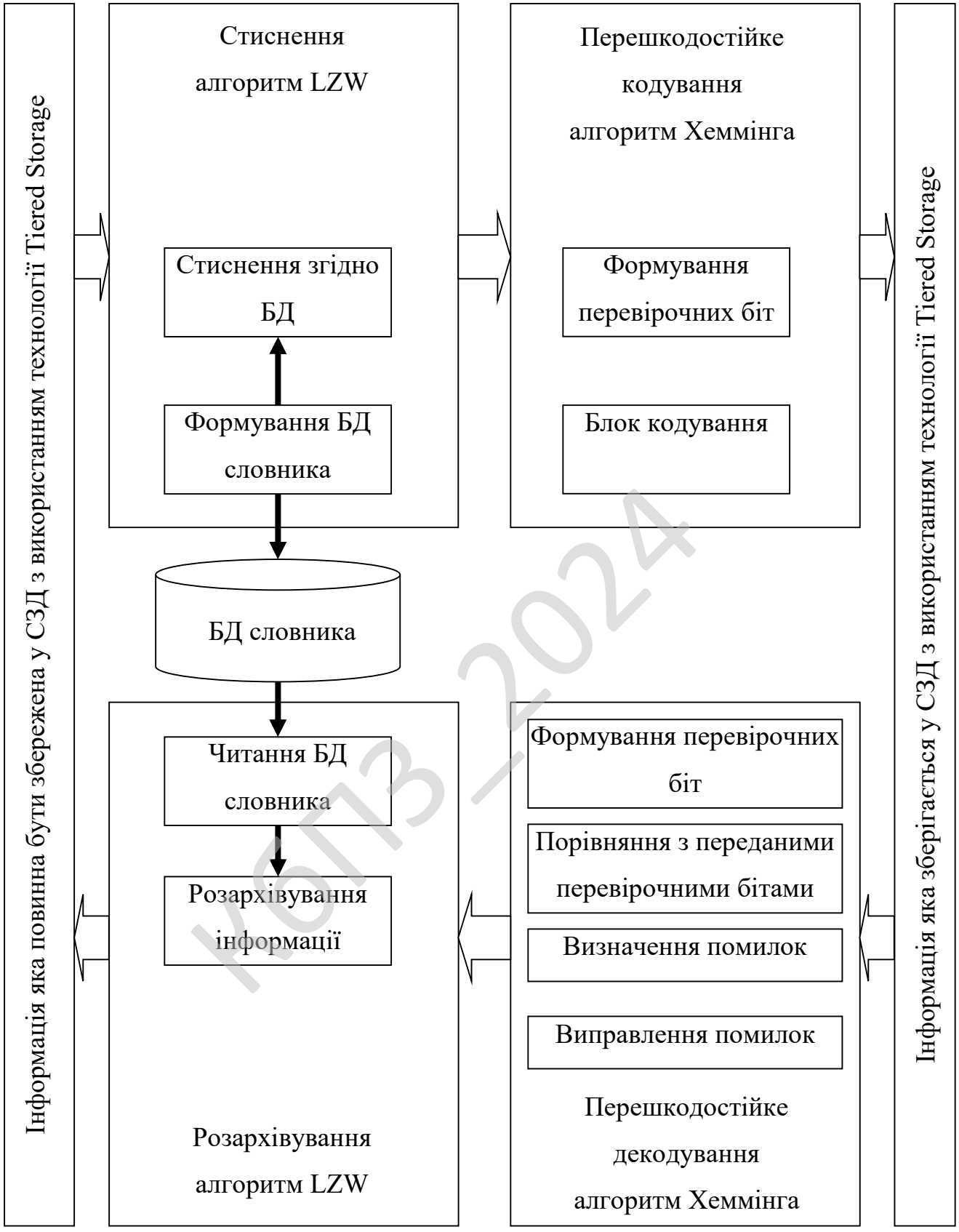

Рисунок 3.1 – Структурна схема системи

3.3 Розробка функціональної схеми

Алгоритм LZW-стиску

Алгоритм LZW-стиску заміняє рядки символів деякими кодами. Це робиться без якого-небудь аналізу вхідного тексту. Замість цього при додаванні кожного нового рядка символів проглядається таблиця рядків. Стиск відбувається, коли код заміняє рядок символів. Коди, генеруємі LZW-алгоритмом, можуть бути будь-якої довжини, але вони повинні містити більше біт, ніж одиничний символ. Перші 256 кодів (коли використовуються 8-бітні символи) за замовчуванням відповідають стандартному набору символів. Інші коди відповідають оброблюваним алгоритмом рядкам.

Стиск

Алгоритм LZW-стиску в найпростішій формі наведений нижче. Щораз, коли генерується новий код, новий рядок додається в таблицю рядків. LZW постійно перевіряє, чи є рядок уже відомим, і , якщо так, виводить існуючий код без генерації нового.

Процедура LZW-стиску:

```
РЯДОК = черговий символ із вхідного потоку
WHILE вхідний потік не порожній DO
    СИМВОЛ = черговий символ із вхідного потоку
    IF РЯДОК+СИМВОЛ у таблиці рядків THEN
        РЯДОК = РЯДОК+СИМВОЛ
    ELSE
        вивести у вихідний потік код для РЯДОК
        додати в таблицю рядків РЯДОК+СИМВОЛ
        РЯДОК = СИМВОЛ
    END of IF
END of WHILE
вивести у вихідний потік код для РЯДОК
```

Простий рядок, використаний для демонстрації алгоритму, наведений в таблиці 3.1. Вхідний рядок є коротким списком англійських слів, розділених символом "/".

Вхідний рядок : /WED/WE/WEE/WEB/WET.

Таблиця 3.1 – Алгоритм стиску

Вхід(символи)	Вихід(коди)	Нові коди й відповідні рядки
/W	/	256 = /W
E	W	257 = WE
D	E	258 = ED
/	D	259 = D/
WE	256	260 = /WE
/	E	261 = E/
WEE	260	262 = /WEE
/W	261	263 = E/W
EB	257	264 = WEB
/	B	265 = B/
WET	260	266 = /WET
<EOF>	T	

Як ви можете помітити, аналізуючи алгоритм, його робота починається з того, що на першому кроці циклу він виконує перевірку на наявність рядка "/W" у таблиці. Коли він не знаходить цей рядок, то генерує код для "/" і додає в таблицю рядок "/W". Т.к. 256 символів уже визначені для кодів 0-255, то першому певному рядку може бути поставлений у відповідність код 256. Після цього система читає наступну букву ("E"), додає другий підрядок ("WE") у таблицю й виводить код для букви "W".

Цей процес повторюється доти, поки другий підрядок, що складається із прочитаних символів "/" і "W", не зіставиться зі строковим номером 256. У цьому випадку система виводить код 256 і додає трьохсимвольний підрядок в таблицю.

Цей процес триває доти, поки не вичерпається вхідний потік і все коди не будуть виведені.

Вихідний потік для заданого рядка показаний у таблиці 3.1, також як і отримана в результаті таблиця рядків. Як ви можете помітити, ця таблиця швидко заповнюється, тому що новий рядок додається в таблицю щораз, коли генерується код. У цьому явно виродженому прикладі було виведено п'ять закодованих підрядків і сім символів. Якщо використовувати 9-бітні коди для виводу, то 19-символьний вхідний рядок буде перетворений в 13.5-символьний вихідний рядок. Звичайно, цей приклад був обраний тільки для демонстрації. У дійсності стиск звичайно не починається доти, поки не буде побудована досить велика таблиця, звичайно після прочитання порядку 100 вхідних байт.

Розпакування

Алгоритму стиску відповідає свій алгоритм розпакування. Він одержує вихідний потік кодів від алгоритму стиску й використовує його для точного відновлення вхідного потоку. Однією із причин ефективності LZW-алгоритму є те, що він не має потреби в зберіганні таблиці рядків, отриманої при стиску. Таблиця може бути точно відновлена при розпакуванні на основі вихідного потоку алгоритму стиску. Це можливо тому, що алгоритм стиску виводить СТРОКОВІ й СИМВОЛЬНІ компоненти коду перш ніж він помістить цей код у вихідний потік. Це означає, що стислі дані не обтяжені необхідністю тягти за собою більшу таблицю перекладу.

Алгоритм розпакування представлений на нижче. Відповідно до алгоритму стиску, він додає новий рядок у таблицю рядків щораз, коли читає із вхідного потоку новий код. Усе, що йому необхідно зробити в добавок – це перевести кожний вхідний код у рядок і переслати її у вихідний потік.

Процедура LZW-розпакування:

читати СТАРИЙ_КОД

вивести СТАРИЙ_КОД

WHILE вхідний потік не порожній DO

читати НОВИЙ_КОД

РЯДОК = перевести НОВИЙ_КОД

вивести РЯДОК

СИМВОЛ = перший символ РЯДКА

додати в таблицю перекладу СТАРИЙ_КОД+СИМВОЛ

СТАРИЙ_КОД = НОВИЙ_КОД

END of WHILE

Таблиця 3.2 – Алгоритм розпакування

Вхід НОВИЙ КОД	СТАРИЙ КОД	РЯДОК Вихід	СИМВОЛ	Новий вхід таблиці
/	/	/		
W	/	W	W	256 = /W
E	W	E	E	257 = WE
D	E	D	D	258 = ED
256	D	/W	/	259 = D/
E	256	E	E	260 = /WE
260	E	/WE	/	261 = E/
261	260	E/	E	262 = /WEE
257	261	WE	W	263 = E/W
B	257	B	B	264 = WEB
260	B	/WE	/	265 = B/
T	260	T	T	266 = /WET

У таблиці 3.2 наведена схема роботи алгоритму на основі стислих даних, отриманих у вище наведеному прикладі. Важливо відзначити, що побудова таблиці рядків алгоритмом розпакування закінчується саме тоді, коли побудована таблиця рядків алгоритму стиску.

Вихідний потік ідентичний вхідному потоку алгоритму стиску. Відзначимо, що перші 256 кодів уже визначені для перекладу одиночних символів, також як і в алгоритмі стиску.

Проблеми

До нещастя алгоритм розпакування, наведений у таблиці 3.2, є все таким занадто простим. В алгоритмі стиску існують деякі виняткові ситуації, які створюють проблеми при розпакуванні. Якщо існує рядок, що представляє пари (РЯДОК СИМВОЛ) і вже певну в таблиці, а вхідний потік, що переглядається, містить послідовність РЯДОК СИМВОЛ РЯДОК СИМВОЛ РЯДОК, алгоритм стиску виведе код перш, ніж розпаковник одержить можливість визначити його.

Простий приклад ілюструє це. Припустимо, рядок "JOEYN" визначений у таблиці з кодом 300. Коли послідовність "JOEYNJOEYNJOEY" з'являється в таблиці, вихідний потік алгоритму стиску виглядає подібно тому, як показано в таблиці 3.3.

Вхідний рядок : ...JOEYNJOEYNJOEY...

Таблиця 3.3 – Кодування

Вхід(символи)	Вихід(коди)	Нові коди й відповідні рядки.
JOEYN	288 = JOEY	300 = JOEYN
A	N	301 = NA
.	.	.
.	.	.
.	.	.
JOEYNJ	300 = JOEYN	400 = JOEYNJ
JOEYNJO	400	401 = JOEYNJO

Коли розпаковник переглядає вхідний потік, він спочатку декодує код 300, потім виводить рядок "JOEYN" і додає визначення для, скажемо, коду 399 у таблицю, хоча він уже міг там бути. Потім читає наступний вхідний код, 400, і

виявляє, що його немає в таблиці. Це вже проблема. На щастя, це відбудеться тільки в тому випадку, якщо розпаковник зустріне невідомий код. Тому що це фактично єдина колізія, те можна без праці вдосконалити алгоритм.

Модифікований алгоритм передбачає спеціальні дії для ще невизначених кодів. У прикладі нижче розпаковник виявляє код 400, що ще не визначений. Тому що цей код не відомий, те декодується значення СТАРОГО_КОДУ, рівне 300. Потім розпаковник додає значення СИМВОЛУ, рівне "J", до рядка. Результатом є правильний переклад коду 400 у рядок "JOEYNJ".

Процедура LZW-розпакування:

```
читати СТАРИЙ_КОД
вивести СТАРИЙ_КОД
СИМВОЛ = СТАРИЙ_КОД
WHILE вхідний потік не порожній DO
    читати НОВИЙ_КОД
    IF NOT у таблиці перекладу НОВИЙ_КОД THEN
        РЯДОК = перевести СТАРИЙ_КОД
        РЯДОК = РЯДОК+СИМВОЛ
    ELSE
        РЯДОК = перевести НОВИЙ_КОД
    END of IF
    вивести РЯДОК
    СИМВОЛ = перший символ РЯДКА
    додати в таблицю перекладу СТАРИЙ_КОД+СИМВОЛ
    СТАРИЙ_КОД = НОВИЙ_КОД
END of WHILE
```

Реалізація

У програмі використовувалися коди довжиною 12, 13 і 14 біт. При довжині коду 12 біт потенційно можливо зберігати до 4096 рядків у таблиці. Щораз, коли читається новий символ, таблиця рядків повинна проглядатися для

зіставлення. Якщо зіставлення не знайдене, новий рядок повинна бути додана в таблицю. Тут виникають дві проблеми. По-перше, таблиця рядків може досить швидко стати дуже великою. Навіть якщо довжина рядків у середньому обмежується 3 або 4 символами кожна, верхня межа довжин рядків може легко перевищити 7 або 8 байт на код. До того ж кількість пам'яті, необхідної для зберігання рядків, заздалегідь не відомо, тому що воно залежить від загальної довжини рядків.

Друга проблема полягає в організації пошуку рядків. Щораз, коли читається новий символ, необхідно організувати пошук для нового рядка виду РЯДОК+СИМВОЛ. Це означає підтримку відсортованого списку рядків. У цьому випадку пошук для кожного рядка включає число порівнянь порядку log2 від загального числа рядків. Використання 12-бітних слів потенційно дозволяє виконувати не більше 12 порівнянь для кожного коду.

Перша проблема може бути вирішена зберіганням рядків як комбінацій код/символ. Тому що кожний рядок у дійсності є поданням комбінації вже існуючого коду й додаткового символу, можна зберігати кожний рядок як окремий код плюс символ. Наприклад у розібраному вище прикладі рядок "/WEE" зберігається як код 260 і символ "E". Це дозволяє використовувати для зберігання тільки 3 байти замість 5 (включающих додатковий байт для кінця рядка). Ідучи назад, можна визначити, що код 260 зберігається як код 256 плюс додатковий символ "E". Нарешті, код 256 зберігається як "/" плюс "W".

Виконання порівняння рядків є небагато більше важким. Новий метод зберігання збільшує час, необхідне для порівняння рядків, але він не впливає на число порівнянь. Ця проблема вирішується використанням алгоритму хешування для зберігання рядків. Це означає, що код 256 не зберігається в якому-небудь масиві за адресою 256, а зберігається в масиві за адресою, сформованому на основі самого рядка. При визначенні місця зберігання даного рядка можна використовувати тестовий рядок для генерації хеш-адреси й потім знайти цільовий рядок однократним порівнянням. Тому що код для будь-якого даного

рядка не можна довідатися надалі інакше як по його позиції в масиві, необхідно зберігати код для даного рядка разом з даними рядка. У демонстраційній програмі для цього використовуються елементи трьох масивів : code_value[i], prefix_code[i] і append_character[i].

Коли необхідно додати новий код у таблицю, використовується хеш-функція в процедурі find_match для генерації коректного і. Процедура find_match генерує адресу й потім перевіряє, чи не використовувався він уже. Якщо це так, то find_match виконує другу пробу й так доти, поки не найдеться вільне місце.

Хеш-функція, використана в цій програмі – проста "xor"-типу хеш-функція. Префікс коду й додатковий символ комбінуються для формування адреси масиву. Якщо вміст префікса коду й символ у масиві зіставляються їм, то вертається коректна адреса. Якщо елемент масиву по цій адресі вже використаний, виконується фіксований зсув для пошуку нового місця. Це виконується доти, поки не буде знайдені вільне місце або не відбудеться зіставлення. Середнє число пошуків у такій таблиці – менше 3, якщо використовується таблиця на 25% більшого розміру, ніж необхідно. Воно може бути поліпшене шляхом збільшення розміру таблиці. Необхідно відзначити, що для того, щоб порядок вторинних проб працював, розмір таблиці повинен бути простим числом. Це пояснюється тим, що проба може бути будь-яким цілим між 1 і розміром таблиці. Якщо проба й розмір таблиці не є взаємно простими, пошук вільних місць може закінчитися невдачею, навіть якщо вони є.

Реалізація алгоритму розпакування має свій набір проблем. Одна із проблем алгоритму стиску тут зникає. Коли виконується стиск, необхідно організувати пошук у таблиці для даного рядка. При розпакуванні необхідно організувати перегляд для окремого коду. Це означає, що можна зберігати префікси кодів і додаткові символи, індексуючись по їхньому строковому коді. Це усуває необхідність у хеш-функції й звільняє масив, що використовувався для зберігання значень кодів.

На жаль метод, використаний для зберігання строкових величин, приводить до того, що декодування рядків повинне виконуватися в інверсному порядку. Це значить, що всі символи для даного рядка при декодуванні повинні міститися в стековий буфер, а потім виводитися у зворотному порядку. У наведеній програмі це виконується функцією decode_string.

Проблема з'являється, коли читання вхідного потоку переривається при досягненні кінця потоку. Для цієї частки випадку в програмі зарезервований останній обумовлений код MAX_VALUE як ознака кінця даних. Це не є необхідним при читанні файлу, але може допомогти при читанні буфера стислих даних з пам'яті. Витрати на втрату одного обумовленого коду досить малі порівняно з усім процесом.

Результати

Досить важко охарактеризувати результативність якої-небудь техніки стиску даних. Ступінь стиску визначається різними факторами. LZW-стиск виділяється серед інших, коли зустрічається з потоком даних, що містять повторювані рядки будь-якої структури. Із цієї причини він працює досить ефективно, коли зустрічає англійський текст. Рівень стиску може досягати 50% і вище. Відповідно, стиск відеоформ і копій екранів показує ще кращі результати.

Труднощі при стиску файлів даних трохи більші. Залежно від даних, результат стиску може бути як гарним, так і не дуже задовільним. У деяких випадках "стислий" файл може перевершувати по своїх розмірах вихідний текст. Невеликий експеримент дасть Вам подання про те, добре або погано впаковуються Ваші дані.

Однією із проблем є те, що наведена програма не адаптується до різної довжини файлів. Використання 14– або 15-бітних кодів дає кращий ступінь стиску на великих файлах (це пояснюється тим, що для них будуються більші таблиці рядків), але гірше працює з маленькими файлами. Такі програми, як "ARC", вирішують цю проблему використанням кодів змінної довжини. Наприклад, коли величина next_code перебуває між 256 і 511, "ARC" читає й

виводить 9-бітні коди. Коли величина next_code стає настільки великий, що необхідно 10-бітні коди, процедури стиску й розпакування збільшують розмір коду. Це значить, що 12– і 15-бітні варіанти програми працюють добре й на маленьких файлах.

Іншою проблемою великих файлів є те, що зі збільшенням числа прочитаних байтів ступінь стиску може почати погіршуватися. Причина проста: тому що розмір таблиці рядків фіксований, після занесення певного числа рядків їх уже просто нікуди додати. Але побудована таблиця потрібна тільки для тої частини файлу, по якій вона була побудована. Частини, що залишилися, файлу можуть мати інші характеристики й у дійсності потрібна вже трохи відмінна таблиця.

Звичайним способом рішення цієї проблеми є контроль ступеня стиску. Після того, як таблиця рядків заповнена, пакувальник стежить за поводженням коефіцієнта стиску. Після певного ступеня його погіршення таблиця рядків очищається й починає будуватися заново. Процедура розпакування визначає цей момент тим, що пакувальник записує у свій вихідний потік спеціальний код. Альтернативним способом є визначення найбільш часте рядків, що зустрічаються, і чищення інших. Адаптивна техніка, подібна цієї, може, однак, зустріти труднощі реалізації в програмах розумного розміру.

І, нарешті, можна брати вироблювані LZW-методом коди й пропускати їх через кодуючий фильтр Хаффмана, що адаптується,. Це дає трохи більший ступінь стиску, але вартість такої роботи більше високий, також як і час обробки.

Для підвищення надійності зберігання даних, при ушкодженні архівного файлу пропонується використовувати алгоритм Хеммінга.

Алгоритм Хеммінга

Коди Хеммінга є кодами, що самоконтролюються, тобто кодами, що дозволяють автоматично виявляти найбільш імовірні помилки при передачі даних. Для їхньої побудови досить приписати до кожного слова один додатковий (контрольний) двійковий розряд і вибрати цифру цього розряду так, щоб загальна

кількість одиниць у зображенні будь-якого числа була, наприклад, парнею. Одиночна помилка в якому-небудь розряді переданого слова (у тому числі, може бути, і в контрольному розряді) змінить парність загальної кількості одиниць. Лічильники за модулем 2, що підраховують кількість одиниць, які втримуються серед двійкових цифр числа, можуть давати сигнал про наявність помилок.

При цьому, зрозуміло, ми не одержуємо ніяких вказівок про те, у якому саме розряді відбулася помилка, і, отже, не маємо можливості виправити неї. Залишаються непоміченими також помилки, що виникають одночасно у двох, у чотирьох або взагалі в парній кількості розрядів. Втім, подвійні, а тим більше чотириразові помилки покладаються малоймовірними.

Коди, що самокоректуються

Коди, у яких можливо автоматичне виправлення помилок, називаються що самокоректуються. Для побудови коду, що самокоректується, розрахованого на виправлення одиночних помилок, одного контрольного розряду недостатньо. Як видно з подальшого, кількість контрольних розрядів k повинне бути обране так, щоб задовольнялося нерівності:

$$2^k \geq k + m + 1,$$

або:

$$k \geq \log_2(k + m + 1),$$

де m – кількість основних двійкових розрядів кодового слова. Мінімальні значення k при заданих значеннях m, знайдені відповідно до цієї нерівності, наведені в таблиці 3.4.

Таблиця 3.4 – Мінімальні значення k при заданих значеннях m

Діапазон m	k_{min}
1	2
2-4	3
5-11	4
12-26	5
27-57	6

Маючи m+k розрядів, що самокоректується код можна побудувати наступним чином.

Привласнимо кожному з розрядів свій номер – від 1 до m+k; запишемо ці номери у двійковій системі числення. Оскільки $2^k > m + k$,кожний номер можна представити, мабуть, k-розрядним двійковим числом.

Припустимо далі, що всі m+k розрядів коду розбиті на контрольні групи, які частково перекриваються, причому так, що одиниці у двійковому поданні номера розряду вказують на його приналежність до певних контрольних груп. Наприклад: розряд № 5 належить до 1-й і 3-й контрольним групам, тому що у двійковому поданні його номера $5_{10} = \ldots 000101_2$ – 1-й і 3-й розряди містять одиниці.

Серед m+k розрядів коду при цьому є k розрядів, кожний з яких належить тільки до однієї контрольної групи:

Розряд № 1: $1_{10} = \ldots 000001_2$ належить тільки до 1-й контрольної групи.

Розряд № 2: $2_{10} = \ldots 000010_2$ належить тільки до 2-й контрольній групі.

Розряд № 4: $4_{10} = \ldots 000100_2$ належить тільки до 3-й контрольній групі.

…

Розряд № 2^{k-1} належить тільки до k-*й* контрольної групи.

Ці k розрядів ми й будемо вважати контрольними. Інші m розрядів, кожний з яких належить, щонайменше, до двох контрольних груп, будуть інформаційними розрядами.

У кожній з k контрольних груп будемо мати по одному контрольному розряді. У кожний з контрольних розрядів помістимо таку цифру (0 або 1), щоб загальна кількість одиниць у його контрольній групі було парним.

Наприклад, досить розповсюджений код Хеминга з m=7 і k=4.

Нехай вихідне слово (кодове слово без контрольних розрядів) – 0110101_2.

Позначимо P_i – контрольний розряд №i; а D_i – інформаційний розряд №i, де i = 1,2,3,4…

Таблиця 3.5 – Параметри й значення коду

№ розряду	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011
Розподіл контрольних і інформаційних розрядів	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7
Інформаційне кодове слово :			0		1	1	0		1	0	1
p_1	1		0		1		0		1		1
p_2		0	0			1	0			0	1
p_3				0	1	1	0				
p_4								0	1	0	1
Кодове слово з контрольними розрядами:	1	0	0	0	1	1	0	0	1	0	1

Цікаво подивитися, як перекриваються контрольні групи в цьому випадку. Перша група контролює розряди № 3,5,7,9,11 вихідні коди, друга – 3,6,7,10,11 (№ групи = № контрольного розряду). Очевидно, що вони частково перекриваються.

Припустимо тепер, що даного кодового слова 10001100101 відбулася помилка в 11 -м розряді, так, що було прийнято нове кодове слово 10001100100. Зробивши в прийнятому коді перевірку парнесті усередині контрольних груп, ми виявили б, що кількість одиниць непарне в 1-й, 2-й і 4-й контрольних групах, і парне в 3-й контрольній групі. Це вказує, по-перше, на наявність помилки, по-друге, означає, що номер помилково прийнятого розряду у двійковому поданні містить одиниці на першому, другому й четвертому місцях і нуль – на третім місці, т.до помилка тільки одна, і 3-я контрольна сума виявилася вірною.

Таблица 3.6 – Параметри коду при помилці

№ розряду	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011		
Розподіл контрольних і інформаційних розрядів	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7	Контроль по парнесті в групі	Контрольний біт
Передане кодове слово:	1	0	0	0	1	1	0	0	1	0	1		
Прийняте кодове слово:	1	0	0	0	1	1	0	0	1	0	0		
p_1	1		0		1		0		1		0	Fail	1
p_2		0	0			1	0			0	0	Fail	1
p_3				0	1	1	0					Pass	0
p_4								0	1	0	0	Fail	1

Таблиця 3.7 – Виявлення помилки

	p_4	p_3	p_2	p_1	
У двійковому поданні	1	0	1	1	
У десятковому поданні	8		2	1	$\Sigma = 11$

З таблиці видно, що помилка відбулася в 11-м розряді і її можна виправити. Побудований код, зрозуміло, не розрахований на можливість одночасної помилки у двох розрядах.

Наприклад, коли помилки одночасно пройшли в 3-м і 7-м розрядах вихідного коду, перші й другий контрольні біти навіть не помітять підміни.

Коли в прийнятому коді виробляється перевірка парнесті усередині контрольних груп, випадок подвійної помилки нічим зовні не відрізняється від випадку одиночної помилки.

Код Хеммінга

Можна побудувати й такий код, що виявляв би подвійні помилки й виправляв одиночні. Для цього до коду, що самокоректується, розрахованому на виправлення одиночних помилок, потрібно приписати ще один контрольний розряд (розряд подвійного контролю). Повна кількість розрядів коду при цьому буде m+k+1. Цифра в розряді подвійного контролю встановлюється такий, щоб загальна кількість одиниць у всіх $m + k + 1$ розрядах коду було парним. Цей розряд не включається в загальну нумерацію й не входить у жодну контрольну групу.

Наприклад, код Хеминга з m=7 і k=4 Нехай інформаційне кодове слово – 0110101

Таблиця 3.8 – Параметри коду

№ розряду	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	Second Parity
Розподіл контрольних і інформаційних розрядів	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7	
Інформаційне кодове слово:			0		1	1	0		1	0	1	
p_1	1		0		1		0		1		1	
p_2		0	0			1	0			0	1	
p_3				0	1	1	0					
p_4								0	1	0	1	
Кодове слово з контрольними розрядами:	1	0	0	0	1	1	0	0	1	0	1	1

При цьому можуть бути наступні випадки.

1. У прийнятому коді в цілому й по всіх контрольних групах кількість одиниць парне. Якщо потрійні помилки й помилки в більшій кількості розрядів виключаються, то перший випадок відповідає безпомилковому прийому коду.

Таблиця 3.9 – Параметри коду при помилці

№ розряду	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	Контроль по парнесті в групі	Контрольний біт	Контроль по парнесті в цілому	Контрольний біт у цілому
Розподіл контрольних і інформаційних розрядів	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7				
Передане кодове слово:	1	0	0	0	1	1	0	0	1	0	1				
Прийняте кодове слово:	1	0	0	0	1	1	0	0	1	0	1				
p_1	1		0		1		0		1		1	Pass	0		
p_2		0	0			1	0			0	1	Pass	0		
p_3				0	1	1	0					Pass	0		
p_4								0	1	0	1	Pass	0	1	Pass

Таблиця 3.10 – Виявлення помилки

	p_4	p_3	p_2	p_1	
У двійковому поданні	0	0	0	0	
У десятковому поданні					$\Sigma = 0$

2. У прийнятому коді в цілому кількість одиниць непарне, але у всіх контрольних групах кількість одиниць парне. Другий випадок – помилки тільки в розряді подвійного контролю.

Таблиця 3.11 – Параметри коду при помилці

№ розряду	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	Контроль по парнесті в групі	Контрольний біт	Контроль по парнесті в цілому	Контрольний біт у цілому
Розподіл контрольних і інформаційних розрядів	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7				
Передане кодове слово:	1	0	0	0	1	1	0	0	1	0	1				
Прийняте кодове слово:	1	0	0	0	1	1	0	0	1	0	1				
p_1	1		0		1		0		1		1	Pass	0		
p_2		0	0			1	0			0	1	Pass	0		
p_3				0	1	1	0					Pass	0		
p_4								0	1	0	1	Pass	0	0	Fail

Таблиця 3.12 – Виявлення помилки

	p_4	p_3	p_2	p_1	
У двійковому поданні	0	0	0	0	
У десятковому поданні					$\Sigma = 0$

3. У прийнятому коді в цілому й у деяких з контрольних груп кількість одиниць непарне. Третій випадок – одиночної помилки в якому-небудь із інших розрядів (можна виправити відповідно до наведеного вище правилами).

Таблиця 3.13 – Параметри коду при помилці

№ розряду	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	Контроль по парності в групі	Контрольний біт	Контроль по парності в цілому	Контрольний біт у цілому
Розподіл контрольних і інформаційних розрядів	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7				
Передане кодове слово :	1	0	0	0	1	1	0	0	1	0	1				
Прийняте кодове слово:	1	0	0	0	1	1	0	0	1	0	0				
p_1	1		0		1		0		1		0	Epic Fail	1		
p_2		0	0			1	0			0	0	Fail	1		
p_3				0	1	1	0					Pass	0		
p_4								0	1	0	0	Fail	1	1	Fail

Таблиця 3.14 – Виявлення помилки

	p_4	p_3	p_2	p_1	
У двійковому поданні	1	0	1	1	
У десятковому поданні	8		2	1	$\Sigma = 11$

З таблиці видно, що помилка відбулася в 11-м розряді й що її можна виправити.

4. У прийнятому коді в цілому кількість одиниць парне, але в деяких контрольних групах є непарна кількість одиниць – подвійна помилка.

Таблиця 3.15 – Параметри коду при помилці

№ розряду	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	Контроль по парності в групі	Контрольний біт	Контроль по парності в цілому	Контрольний біт у цілому
Розподіл контрольних і інформаційних розрядів	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7				
Передане кодове слово:	1	0	0	0	1	1	0	0	1	0	1				
Прийняте кодове слово:	1	0	1	0	1	0	0	0	1	0	1				
p_1	1		1		1		0		1		1	Fail	1		
p_2		0	1			0	0			0	1	Pass	0		
p_3				0	1	0	0					Fail	1		
p_4								0	1	0	1	Pass	0	1	Pass

Таблиця 3.16 – Виявлення помилки

	p_4	p_3	p_2	p_1	
У двійковому поданні	0	1	0	1	
У десятковому поданні		4		1	$\Sigma = 5$

Раз сума, що вийшла, не дорівнює нулю, а контрольний біт указує на помилку передачі, те виявляємо подвійну помилку. Виправлення подвійних помилок тут, звичайно, неможливо.

Збільшуючи далі кількість контрольних розрядів, можна було б побудувати коди, розраховані на виправлення подвійних помилок і виявлення потрійних і т.д. Однак методи побудови цих кодів не цілком розроблені.

Алгоритм LZW

– Ініціалізація словника всіма можливими односимвольними фразами. Ініціалізація вхідної фрази w першим символом повідомлення.

– Зчитати черговий символ K з кодуємого повідомлення.

– Якщо КІНЕЦЬ_ПОВІДОМЛЕННЯ, то видати код для w, інакше

– Якщо фраза w уже є в словнику, привласнити вхідній фразі значення w і перейти до Кроку 2, інакше видати код w, додати w у словник, привласнити вхідній фразі значення K і перейти до Кроку 2.

Кінець.

Алгоритм Хеммінга

Алгоритм Хеммінга діє наступними чином.

Код Хеммінга являє собою блоковий код, що дозволяє виявити й виправити помилково переданий біт у межах переданого блоку. Звичайно код Хеммінга характеризується двома цілими числами, наприклад, (11,7) використовуваний при передачі 7-бітних ASCII-кодів. Такий запис говорить, що при передачі 7-бітного коду використовується 4 контрольних біта (7+4=11). При цьому передбачається, що мала місце помилка в одному біті й що помилка у двох або більше бітах істотно менш імовірна. З обліком цього виправлення помилки здійснюється з певною ймовірністю. Наприклад, нехай можливі наступні правильні коди (всі вони, крім перш і останнього, відстоять друг від друга на відстань 4):

00000000

11110000

00001111

11111111

При одержанні коду 00000111 не важко припустити, що правильне значення отриманого коду дорівнює 00001111. Інші коди відстоять від отриманого на більшу відстань Хеммінга.

Розглянемо приклад передачі коду букви s = 0x073 = 1110011 з використанням коду Хеммінга (11,7).

Таблиця 3.17 – Структура повідомлення по коду Хеммінга

Позиція біта:	11	10	9	8	7	6	5	4	3	2	1
Значення біта:	1	1	1	*	0	0	1	*	1	*	*

Символами * позначені чотири позиції, де повинні розміщатися контрольні біти. Ці позиції визначаються цілим ступенем 2 (1, 2, 4, 8 і т.д.). Контрольна сума формується шляхом виконання операції XOR (виключаєче АБО) над кодами позицій ненульових біт. У цьому випадку це 11, 10, 9, 5 і 3. Обчислимо контрольну суму.

Таблиця 3.18 – Контрольна сума

11 =	1011
10 =	1010
09 =	1001
05 =	0101
03 =	0011
S =	1110

Таким чином, приймач одержить наступний код.

Таблиця 3.19 – Отриманий код

Позиція біта:	11	10	9	8	7	6	5	4	3	2	1
Значення біта:	1	1	1	1	0	0	1	1	1	1	0

Просумуємо знову коди позицій ненульових біт і одержимо нуль.

Таблиця 3.20 – Сума кодів позицій ненульових біт

11 =	1011
10 =	1010
09 =	1001
08 =	1000
05 =	0101
04 =	0100
03 =	0011
02 =	0010
S =	0000

Ну а тепер розглянемо два випадки помилок в одному з біт посилки, наприклад, у біті 7 (1 замість 0) і в біті 5 (0 замість 1). Просумуємо коди позицій ненульових біт ще раз.

Таблиця 3.21 – Сума кодів позицій ненульових біт

11 =	1011
10 =	1010
09 =	1001
08 =	1000
07 =	0111
05 =	0101
04 =	0100
03 =	0011
02 =	0010
S =	0111

11 =	1011
10 =	1010
09 =	1001
08 =	1000
04 =	0100
03 =	0011
02 =	0010
S =	0001

В обох випадках контрольна сума дорівнює позиції біта, переданого з помилкою. Тепер для виправлення помилки досить інвертувати біт, номер якого

зазначений у контрольній сумі. Зрозуміло, що якщо помилка відбудеться при передачі більш ніж одного біта, код Хеммінга при даній надмірності виявиться марний.

У загальному випадку код має N=M+C біт і передбачається, що не більш ніж один біт у коді може мати помилку. Тоді можливо N+1 стан коду (правильний стан і n помилкових). Нехай M=4, а N=7, тоді слово-повідомлення буде мати вигляд: M4, M3, M2, C3, M1, C2, C1. Тепер спробуємо обчислити значення C1, C2, C3. Для цього використовуються рівняння, де всі операції являють собою додавання за модулем 2:

$$C1 = M1 + M2 + M4$$

$$C2 = M1 + M3 + M4$$

$$C3 = M2 + M3 + M4$$

Для визначення того, чи доставлене повідомлення без помилок, обчислюємо наступні вираження (додавання за модулем 2):

$$C11 = C1 + M4 + M2 + M1$$

$$C12 = C2 + M4 + M3 + M1$$

$$C13 = C3 + M4 + M3 + M2$$

Результат обчислення інтерпретується в такий спосіб.

Таблиця 3.22 – Результат обчислення

C11	C12	C13	Значення
1	2	4	Позиція біт
0	0	0	Помилок немає
0	0	1	Біт C3 не вірний
0	1	0	Біт C2 не вірний
0	1	1	Біт M3 не вірний
1	0	0	Біт C1 не вірний
1	0	1	Біт M2 не вірний
1	1	0	Біт M1 не вірний
1	1	1	Біт M4 не вірний

Описана схема легко переноситься на будь-яке число n і M.

Число можливих кодових комбінацій M завадостійкого коду ділиться на n класів, де N – число дозволених кодів. Поділ на класи здійснюється так, щоб у кожний клас увійшов один дозволений код і найближчі до нього (по відстані Хеммінга) заборонені коди. У процесі прийому даних визначається, до якого класу належить код, що прийшов. Якщо код прийнятий з помилкою, він заміняється найближчим дозволеним кодом. При цьому передбачається, що кратність помилки не більше q_m.

Можна довести, що для виправлення помилок із кратністю не більше $q_m m$ (як правило, воно вибирається рівним $D = 2q_m + 1$). У теорії кодування існують наступні оцінки максимального числа N n-розрядних кодів з відстанню D.

Таблиця 3.23 – Визначення кількості помилок в залежності від кодової відстані

d=1	$n=2^n$
d=2	$n=2\ n^{-1}$
d=3	$n\ 2^n/(1+n)$
d=2q+1	$N \le 2^n (1 + \sum_{i=1}^{d} C_n^i)^{-1}$ (для коду Хеммінга ця нерівність перетворюється в рівність)

У випадку коду Хеммінга перші k розрядів використовуються в якості інформаційних, причому $k = n - \log(n+1)$, звідки випливає (логарифм по підставі 2), що k може приймати значення 0, 1, 4, 11, 26, 57 і т.д., це й визначає відповідні коди Хеммінга (3,1); (7,4); (15,11); (31,26); (63,57) і т.д.

На рисунку 3.2 зображена функціональна схема системи. Нижче розглянемо її більш докладно.

Основними функціями розробленого програмного забезпечення є наступні:

– Вибір ступеню стиску файлів.

– Блок формування багатотомних архівів.
– Блок добування інформації з багатотомних архівів.
– Блок формування архівів, що саморозпаковуються.
– Вибір режиму solid.
– Блок виводу моніторингу роботи архіватора.
– Блок вибору способу керування програмою.
– Блок вибору мови інтерфейсу.
– Блок допомоги.

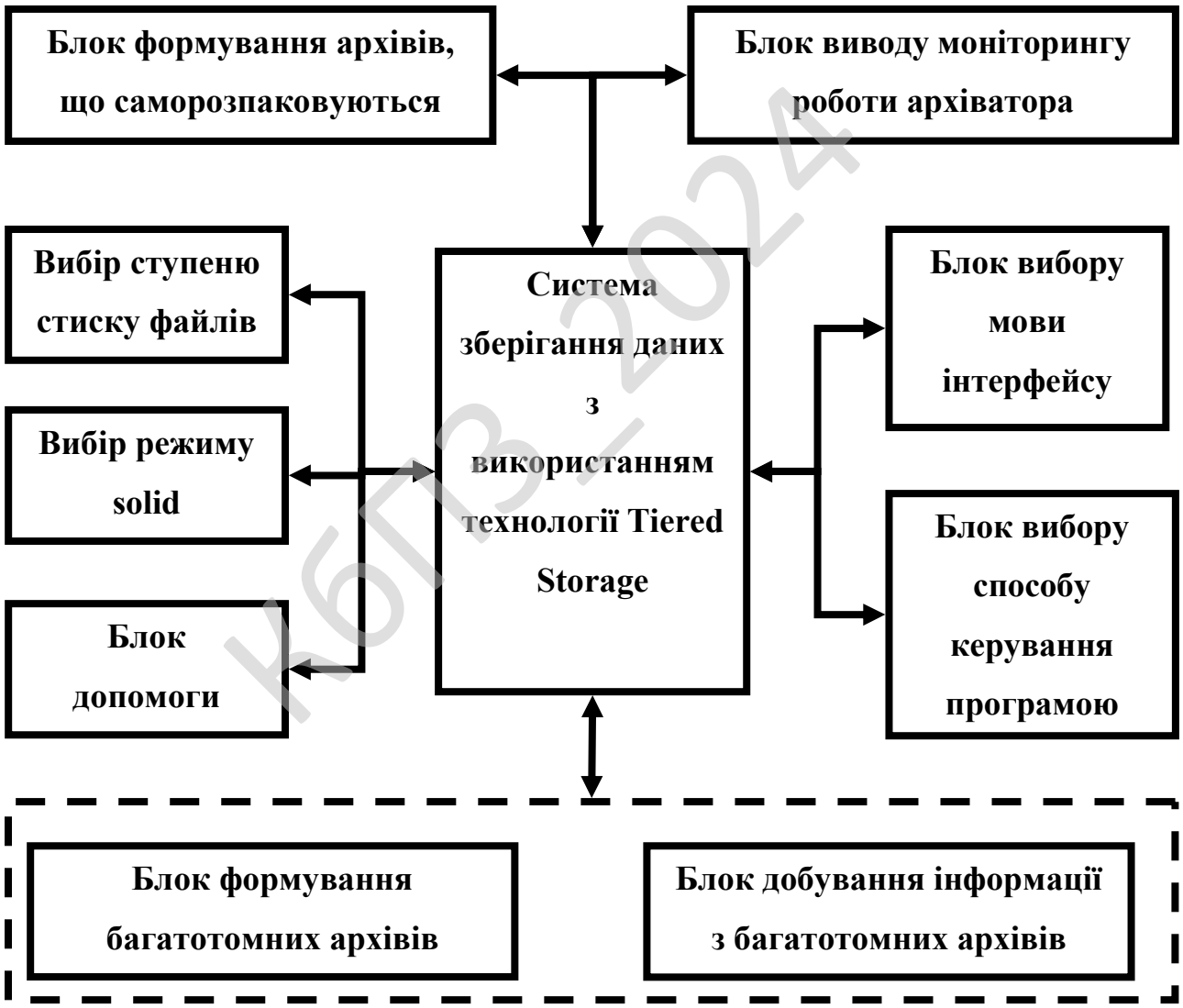

Рисунок 3.2 – Функціональна схема розробленої системи

Розглянемо ці блоки більш детальніше.

Великі за обсягом архівні файли можуть бути розміщені на декількох дисках (томах). Такі архіви називаються багатотомними. Том – це складова частина багатотомного архіву.

Програми-архіватори дозволяють створювати й такі архіви, для добування з яких файлів, що втримуються в них, не потрібні які-небудь програми, тому що самі архівні файли можуть містити програму розпакування. Такі архівні файли називаються такими, що саморозпаковуються.

Архівний файл, що саморозпаковується, – це завантажувальний модуль, що виконується, який здатний до самостійного розархівації файлів, що перебувають у ньому, без використання програми-архіватора. Архів, що саморозпаковується, одержав назву SFX-архів (Self-eXtracting). Архіви такого типу в MS DOS звичайно створюються у формі .exe-файлу. Багато програм-архіваторів роблять розпакування файлів, вивантажуючи їх на диск, але є й такі, які призначені для створення впакованого модуля, що виконується, (програми). У результаті такого впакування створюється програмний файл із тими ж ім'ям і розширенням, що при завантаженні в оперативну пам'ять саморозпаковується й відразу запускається. Разом з тим можливо й зворотне перетворення програмного файлу в розпакований формат. Програми-архіватори крім звичайного режиму стиску, мають режим solid, у якому створюються архіви з підвищеним ступенем стиску й особою структурою організації. У таких архівах всі файли стискуються як один потік даних, тобто областю пошуку повторюваних послідовностей символів є вся сукупність файлів, завантажених в архів, і тому розпакування кожного файлу, якщо він не перший, пов'язана з обробкою інших. Архіви такого типу переважніше використовувати для архівування великої кількості однотипних файлів.

Способи керування програмою-архіватором

Керування програмою-архіватором здійснюється одним із двох способів:

– за допомогою командного рядка MS DOS, у якій формується команда

запуску, що містить ім'я програми-архіватора, команду керування й ключі її настроювання, а також імена архівного й вихідного файлів;

– за допомогою убудованої оболонки й діалогових панелей, що з'являються після запуску програми й дозволяють вести керування з використанням меню й функціональних клавіш, що створює для користувача більше комфортні умови роботи.

Виконуючи запропоновані їй дії, програма-архіватор, як правило, виводить на екран протокол своєї роботи. Всі сучасні програми-архіватори оснащені екранами допомоги, які викликаються при уведенні в командному рядку тільки одного ім'я програми або ім'я із ключем /?. Допомога може бути короткої – на одному екрані або розгорнутої – на декількох. Багато хто архіватори мають екрани допомоги із прикладами складання команд для виконання різних операцій. Інформація допомоги звичайно виводиться на англійській або іншій міжнародній мові.

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

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання бакалаврської дипломної роботи, наведена 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.

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.

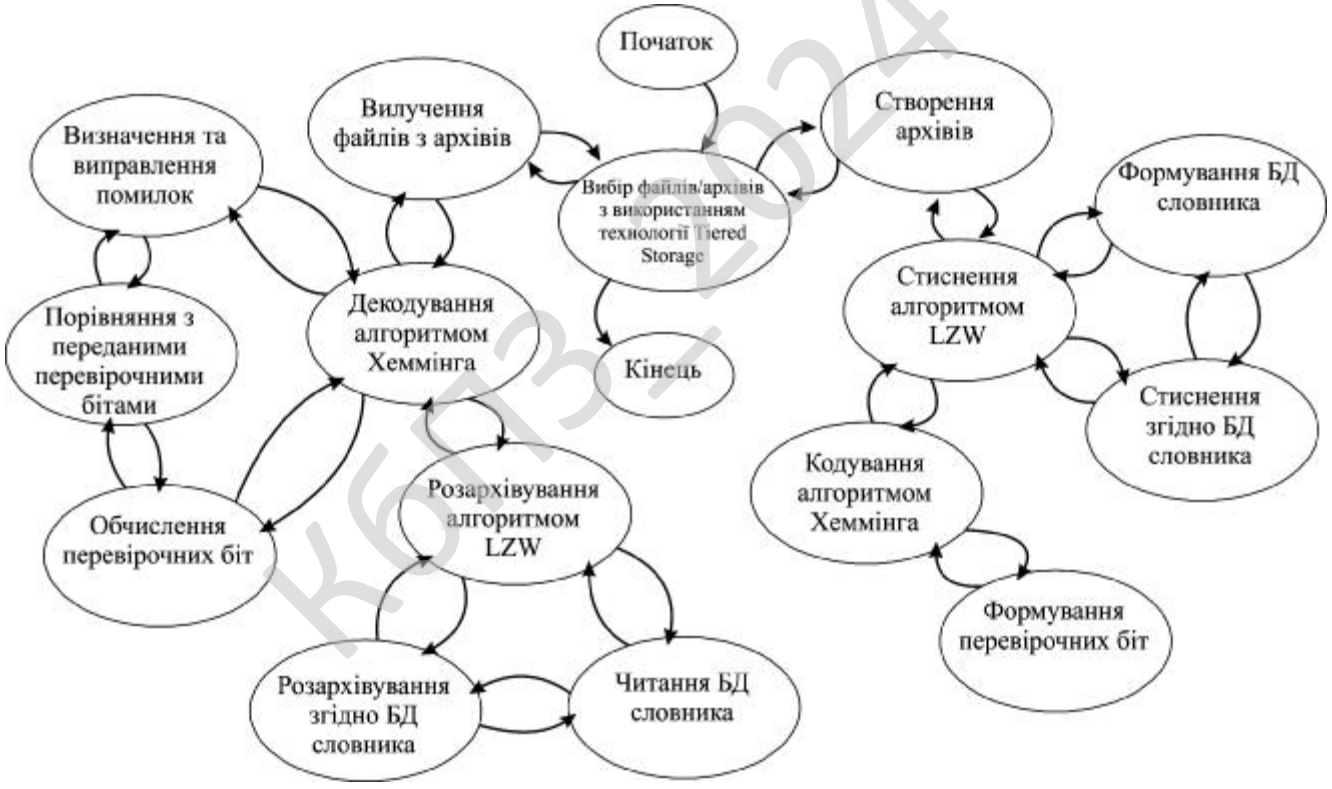

Рисунок 3.3 – Діаграма взаємодії процесів

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

Первинною стадією без якої не відбувається розробка програмного забезпечення це звичайно розробка блок-схем. 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.

Опис алгоритмів функціонування системи

Блок-схеми є першоджерелами стратегії розвитку ПЗ. Тому від точності і детальної блок-схеми залежить результат всієї програми. При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації програми високого рівня, також те, що при розробці програми слід надати особливу увагу модулю зберігання даних з використанням технології Tiered Storage. При складанні блок-схем програмного забезпечення і напрацювання алгоритмів я зіткнувся з масою проблем, які вимагали напрацювання процедур і функцій над основною проблематикою. Для чого були створені додаткові класи, типи даних і константи, що забезпечило вирішення проблем.

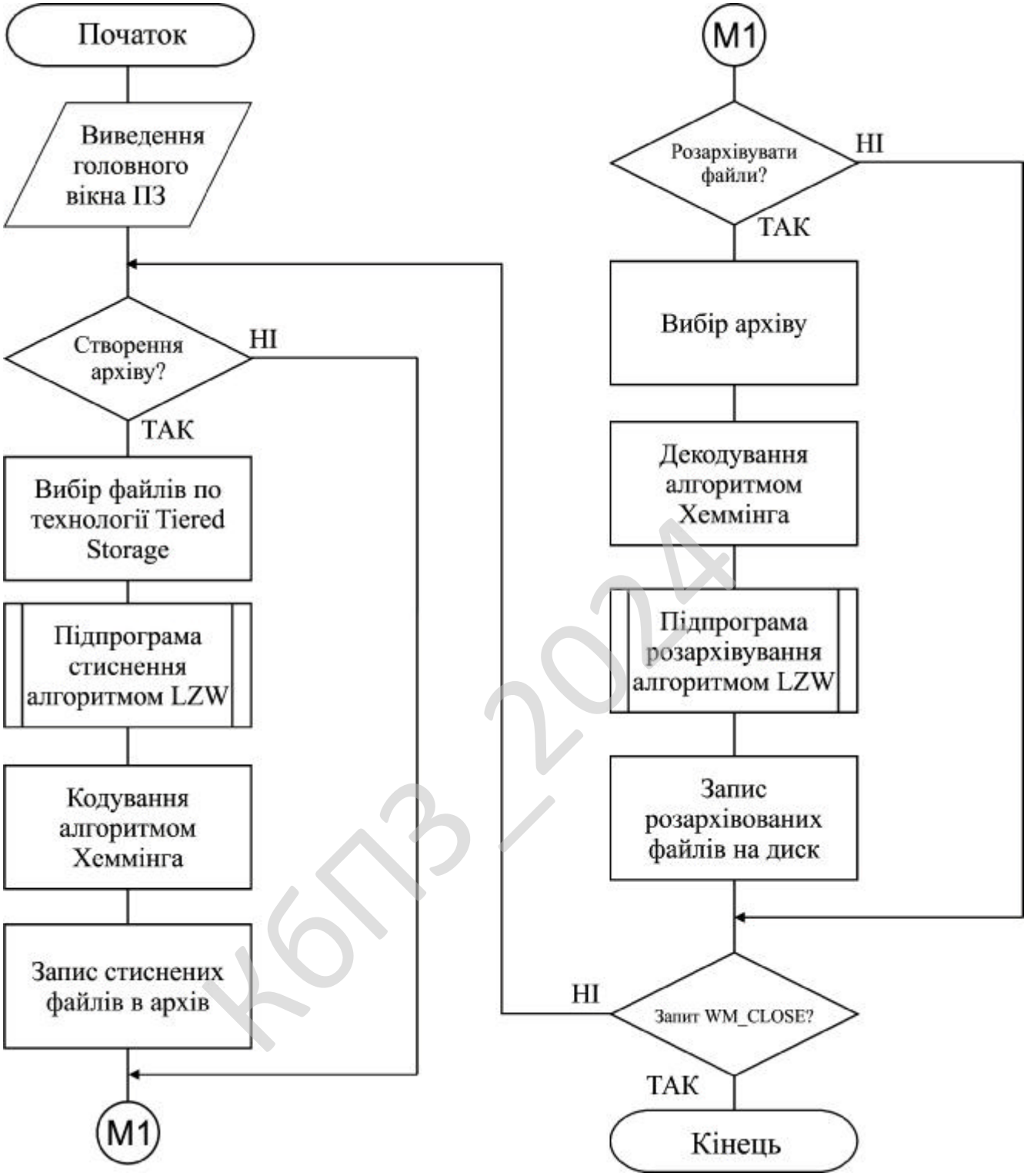

Рисунок 4.1 – Блок-схема основної програми

Організація запису й читання файлу

Перш ніж приступитися до реалізації алгоритмів зберігання даних з використанням технології Tiered Storage варто подумати про спосіб збереження

даних на диску (і читання з диска). Стандартними засобами будь-якої мови програмування можно здійснювати читання-запис файлу тільки по байтах (я маю на увазі мінімально можливий тип даних). Для написання ПЗ нам необхідно мати можливість читати й записувати дані по бітах.

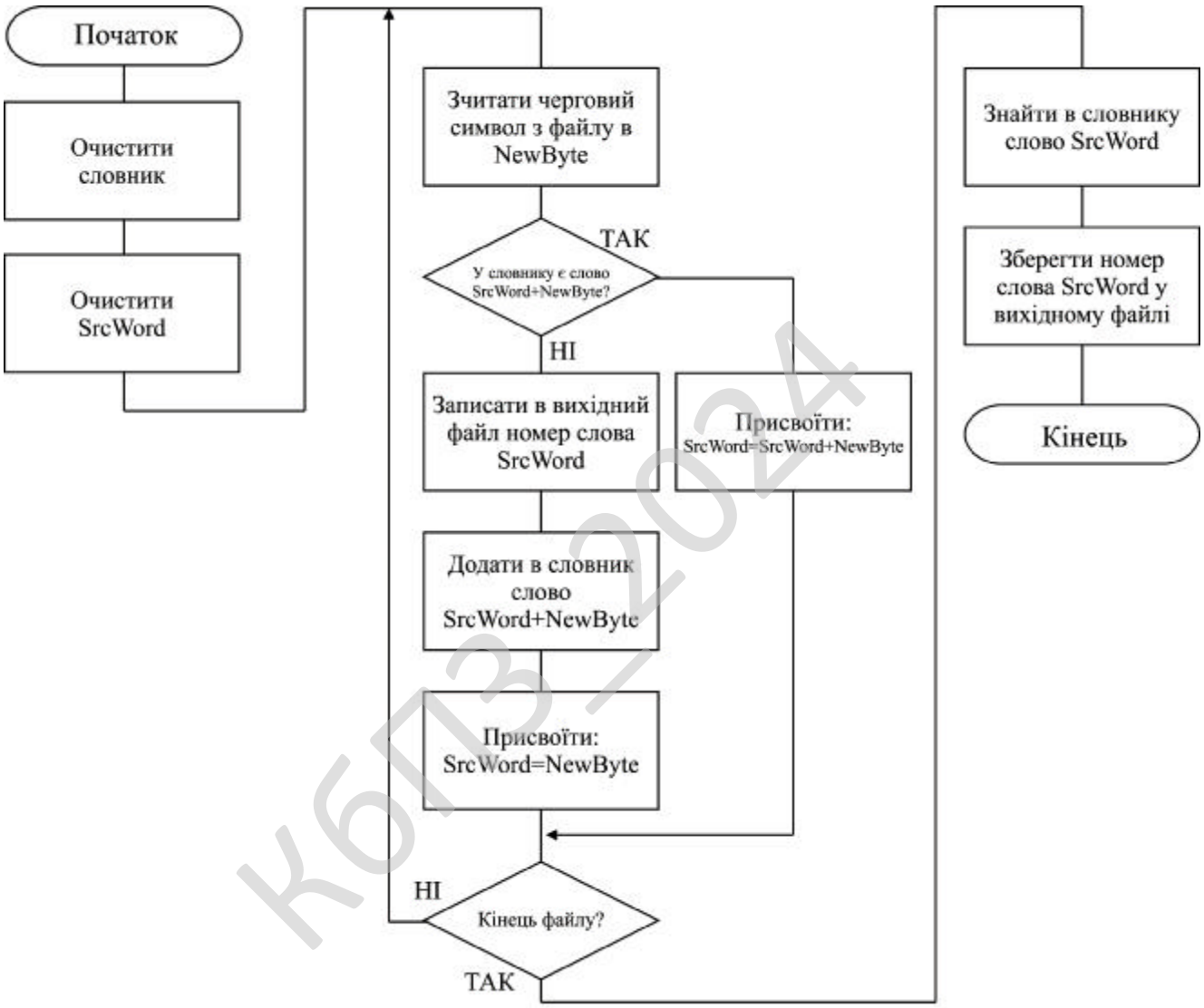

Рисунок 4.2 – Блок-схема роботи підпрограми

Для даної системи мною був написаний модуль DFileStream. Він дозволяє як зчитувати, так і записувати дані у файл ланцюжками біт, довжиною від 1 до 16 біт за раз. Т.я. завдання даного матеріалу – це створення тільки ядра системи, те

реалізація модуля DFileStream виходять за рамки викладу. Нижче будуть описані тільки необхідні функції з нього і їхнє призначення.

Опис функції побітного запису.

```
TFileWriteStream            //клас для організації побітного запису
  OpenStream(FileName:String)    //відкрити файл FileName для запису
  CloseStream          //закрити потік
  WriteBit(BitValue:Byte)   //записати у файл 1 (один) біт
  Write(BitMap:Word; BitsCount:Byte)
//записати у файл декілька біт (BitsCount), BitMap – бітова маска
  BytesWrote           //кількість записаних байт
TFileReadStream             //клас для організації побітного читання
  OpenStream(FileName:String)    //відкрити файл FileName для читання
  CloseStream          //закрити потік
  ReadBit              //завантажити з файлу 1 (один) біт
  Read(BitsCount:Byte)      //завантажити з файлу декілька біт (BitsCount)
  FileRead                  // чи перевіряє прочитаний файл повністю
```

Перераховані вище процедури й функції придадуться нам при реалізації алгоритмів стиску й розпакування файлів.

Словник

Вище вже були перераховані вимоги, пропоновані до словника. Для організації словника я написав модуль Dictionary, у якому описаний клас TDictionary. Нижче наведені необхідні функції даного класу.

```
TDictionary     //клас, що описує словник
Init            //конструктор
Done            //деструктор
Clear           //очистити словник, і проініціалізувати його
Add(SingleWord:TSingleWord)  //додати слово в словник
Find(SingleWord:TSingleWord)
//знайти слово в словнику, якщо слова ні, те вернеться значення "-1"
Words           //Масив всіх слів у словнику
Count           //кількість слів у словнику
```

Як тип TSingleWord використовується стандартний тип «ANSIString Type TSingleWord=ANSIString;». Після створення або очищення словника, перших 256 слів його заповнюються символами з таблиці ASCII.

Програмна реалізація алгоритмів

Самим серцем зберігання даних з використанням технології Tiered Storage є функції стиску й розпакування файлів, тому постараємося написати їх (дотримуючись словесних алгоритмів). Весь інший програмний код можна буде наростити самостійно, якщо звичайно мати деякі пізнання в програмуванні.

Стиск файлу

Перед викликом функції стиску необхідно створити два бітових файлових потоки:

– для читання – вихідний файл;

– для запису – створюваний (вихідний) файл.

Також перед безпосереднім стиском необхідно у вихідному файлі зберегти ім'я й розмір вихідного файлу в прямому виді (не стислому). Це необхідно для наступного розпакування файлу.

Тепер необхідно встановити правила для формату даних у стислому файлі. Виділимо для номера словника 12 біт – одержимо 4096 можливих номерів (від 0 до 4095). Це значить, що словник може містити не більше 4096 слів. Для зберігання ж просто одного символу потрібно 8 біт.

Установимо наступні правила:

– якщо ми записуємо у вихідний файл просто один символ, то спочатку запишемо один біт, дорівнює нулю (“0”). Потім записуємо сам символ (тобто 8 його біт);

– якщо ми записуємо у вихідний файл номер слова зі словника, то спочатку запишемо один байт, дорівнює одиниці (“1”). Потім записуємо номер слова зі словника (тобто 12 його біт).

Все це нам необхідно, щоб при розпакуванні розрізнити, де в нас просто байт, а де номер слова.

А тепер напишемо код, дотримуючись словесного алгоритму.

Примітка: у процедурі використовуються глобальні змінні Dic:TDictionary; ReadStream:TFileReadStream; WriteStream:TfileWriteStream. Також викликається

процедура запису номера слова у вихідний файл WriteByDic (вона буде розглянута нижче).

```
Procedure ProcessCompression;
Var
  SrcWord:TSingleWord;
  NewByte:Byte;
Begin
  Dic.Clear;
//очистимо словник
  SrcWord:='';
  While Not(ReadStream.FileRead) Do
//поки вихідний Файл не зчитаний
  Begin
    NewByte:=ReadStream.Read(8);
//завантажуємо 1 байт
    If Dic.Find(SrcWord+Chr(NewByte)) Then
//якщо слово є в словнику
      SrcWord:=SrcWord+Chr(NewByte)
//то змінимо слово SrcWord
    Else Begin
//інакше
      WriteByDic(Dic.Find(SrcWord));
//запишемо у вихідний файл
      Dic.Add(SrcWord+Chr(NewByte));
//додамо в словник
      SrcWord:=Chr(NewByte);
    End;
  End;
  WriteByDic(Dic.Find(SrcWord));
```

Процедура WriteByDic зберігає у вихідному файлі або символ, або номер слова, залежно від переданого в неї номера.

```
Procedure WriteByDic(N:Integer);
Var
  Data:Word;
Begin
  If N<0 Then Exit;
  If N<256 Then
//якщо це просто символ
  Begin
```

```
    WriteStream.WriteBit(0);
    WriteStream.Write(N, 8);
  End
  Else Begin  //якщо це слово
    WriteStream.WriteBit(1);
    WriteStream.Write(N, 12);
  End; End;
```

Розпакування файлу

Ініціалізуючі дії перед розпакуванням файлу аналогічні діям при запакуванні. Також необхідно вважати розмір вихідного файлу (помнете ми його зберігали) у змінну FSize.

Але от сам алгоритм буде більш складний (більше громіздкий).

```
Procedure ProcessDeCompression;
Var
  SrcWord, SaveWord:TSingleWord;
  i, N, NewCode:Integer;
  NewByte:Byte;
Begin
  Dic.Clear;
  SrcWord:='';
  While WriteStream.BytesWrote<Fsize Do
//поки не збережений весь файл
  Begin
    If ReadStream.ReadBit=0 Then
    Begin
//якщо це просто символ
      NewByte:=ReadStream.Read(8);
      WriteStream.Write(NewByte, 8);
//перепишемо його на вихід
      If Dic.Find(SrcWord+Chr(NewByte)) Then
//якщо слово є в словнику
        SrcWord:=SrcWord+Chr(NewByte)
//те змінимо слово SrcWord
      Else Begin
//інакше
        Dic.Add(SrcWord+Chr(NewByte));
//додамо в словник
        SrcWord:=Chr(NewByte);
      End;
    End
```

```
    Else Begin
//якщо це номер слова
      NewCode:=ReadStream.Read(12);
      If NewCode>=Dic.Count Then
//якщо слова ще немає в словнику
      Begin
        N:=2;
        NewByte:=Ord(SrcWord[1]);
        If Dic.Find(SrcWord+Chr(NewByte)) Then
//якщо слово є в словнику
          SrcWord:=SrcWord+Chr(NewByte)
//те змінимо слово SrcWord
        Else Begin
//інакше
          Dic.Add(SrcWord+Chr(NewByte));
//додамо в словник
          SrcWord:=Chr(NewByte);
        End;
      End
      Else
        N:=1;
      End;
      SaveWord:=Dic.Words[NewCode];
      For i:=1 To Length(SaveWord) Do
//збережемо слово
        WriteStream.Write(SaveWord[i], 8);
      For i:=N To Length(SaveWord) Do
      Begin
        NewByte:=Ord(SaveByte[i]);
        If Dic.Find(SrcWord+Chr(NewByte)) Then
//якщо слово є в словнику
          SrcWord:=SrcWord+Chr(NewByte)
//те змінимо слово SrcWord
        Else Begin
//інакше
          Dic.Add(SrcWord+Chr(NewByte));   //додамо в словник
          SrcWord:=Chr(NewByte);
        End;
      End;
    End;
  End;
End;
```

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

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.

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.

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.

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою алгоритму FEAL – блоковий шифр, запропонований Акіхіро Симідзу і Седзі Міягуті.

У ньому використовуються 64-бітовий блок і 64-бітовий ключ. Його ідея полягає і в тому, щоб створити алгоритм, подібний DES, але з більш сильною функцією етапу. Використовуючи менше етапів, цей алгоритм міг би працювати швидше. На жаль, дійсність виявилася далекою від цілей проекту.

Як вхід процесу шифрування використовується 64-бітовий блок відкритого тексту. Спочатку блок даних підлягає операції XOR з 64 бітами ключа. Потім блок даних розщеплюється на ліву і праву половини. Об'єднання лівої і правої половин за допомогою XOR утворює нову праву половину. Ліва половина і нова права половина проходять через N етапів (спочатку 4). На кожному етапі половина об'єднується за допомогою функції F[1] з 16 бітами ключа і за допомогою XOR – з лівою половиною, створюючи нову праву половину. Вихідна права половина (на початок етапу) стає новою лівою половиною. Після N етапів (ліва і права половини не переставляти після N-го етапу) ліва половина знову об'єднується з допомогою XOR з правою половиною, утворюючи нову праву половину, потім ліва і права об'єднуються разом в 64-бітове ціле. Блок даних об'єднується за допомогою XOR з іншими 64 бітами ключа і алгоритм завершується.

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 зберігання даних з використанням технології Tiered Storage складається з наступних функціональних блоків:

– Навігаційне меню: Файл; Параметри; Довідка.

– Функціональних кнопок ПЗ: Оновлення; Архівувати; Розархівувати; Довідка.

– Обрання диску перегляду поточних даних.

– Розділу відображення файлів обраного файлового каталогу на диску.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

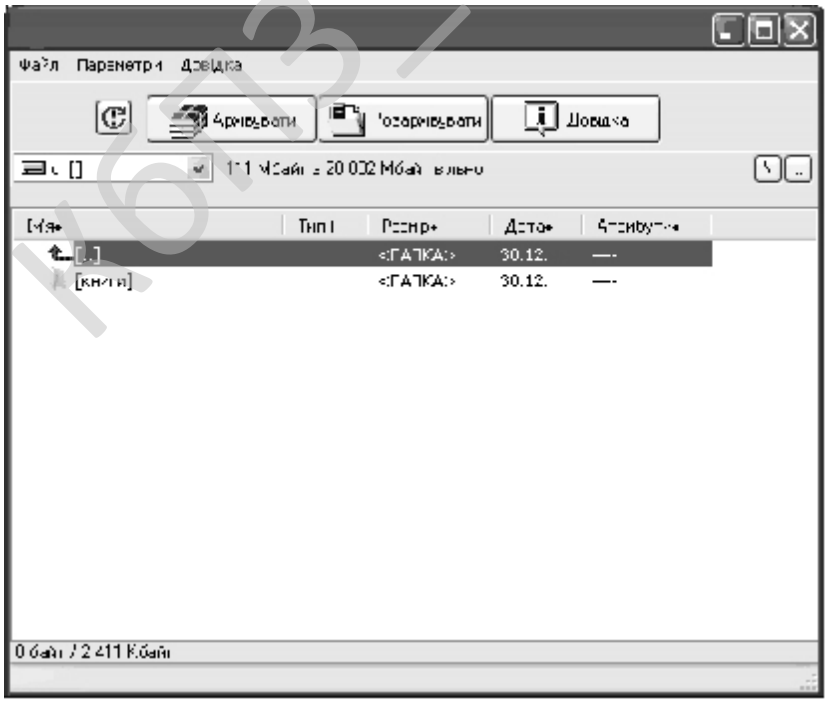

Рисунок 5.1 – Головне вікно розробленого ПЗ

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.

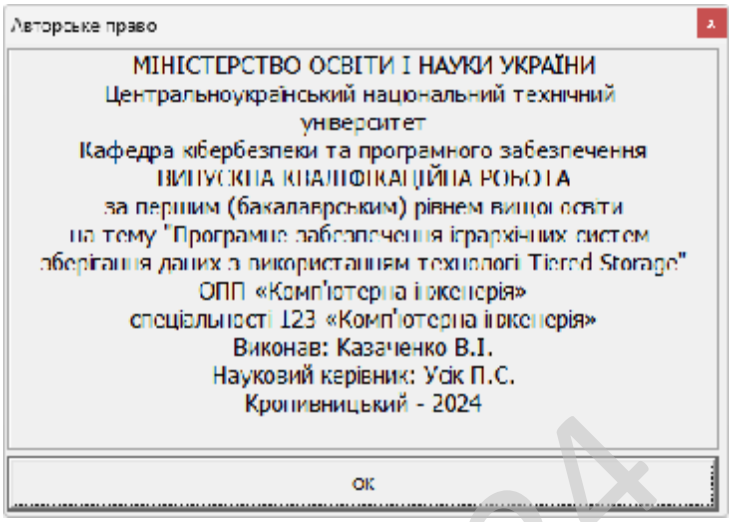

Рисунок 5.2 – Вікно розробника ПЗ

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 білої скриньки та чорної скриньки.

Тестування форматом білої скриньки засноване на аналізі керуючої структури програми. Програма вважається повністю перевіреною, якщо проведено вичерпне тестування маршрутів (шляхів) її графа управління.

У цьому випадку формуються тестові варіанти, в яких:

– Гарантується перевірка всіх незалежних маршрутів програми.

– Знаходяться гілки True, False для всіх логічних рішень.

– Виконуються всі цикли (у межах їхніх кордонів та діапазонів).

– Аналізується правильність внутрішніх структур даних.

Недоліки тестування "білої скриньки":

– Кількість незалежних маршрутів може бути дуже велика.

– Повне тестування маршрутів не гарантує відповідності програми вихідним вимогам до неї.

– У програмі можуть бути пропущені деякі маршрути.

– Не можна виявити помилки, поява яких залежить від даних.

Переваги тестування "білої скриньки" пов'язані з тим, що принцип «білої скриньки» дозволяє врахувати особливості програмних помилок:

– Кількість помилок мінімально в «центрі» і максимально на «периферії» програми.

– Попередні припущення про ймовірність потоку керування або даних у програмі часто бувають некоректними. У результаті типовим може стати маршрут, модель обчислень за яким опрацьована слабо.

– При записі алгоритму програмного забезпечення у вигляді тексту на мові програмування можливе внесення типових помилок трансляції (синтаксичних та семантичних).

– Деякі результати в програмі залежать не від вихідних даних, а від внутрішніх станів програми.

Проводилось тестування 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 ієрархічних систем зберігання даних з використанням технології Tiered Storage.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем зберігання даних з використанням технології Tiered Storage.

– Досліджена система зберігання даних з використанням технології Tiered Storage.

– На основі отриманих результатів досліджень створена програмна реалізація ієрархічних систем зберігання даних з використанням технології Tiered Storage.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання зберігання даних з використанням технології Tiered Storage.

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 Delphi 10.4 Sydney. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для ієрархічних систем зберігання даних з використанням технології Tiered Storage.

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 Feal.

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

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

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

4. Kuznetsov, O., Kuznetsova, Y., Smirnov, O., Kostenko, O., Zvieriev, V. «Evaluating Hashing Algorithms in the Age of ASIC Resistance». *CEUR Workshop Proceedings*, 2023, 3628, pp. 93-105.

5. Kuznetsov O., Frontoni E., Kuznetsova Ye., Smirnov O., Chevardin V. «Achieving Enhanced Security in Biometric Authentication: A Rigorous Analysis of Code-Based Fuzzy Extractor». *CEUR Workshop Proceedings,* Volume 3624, 2023, pp. 330-339.

6. 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.

7. 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.

8. 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,

9. 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.

10. 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.

11. 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

12. 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.

13. 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.

14. 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.*

15. 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.

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

17. 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.

18. 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.

19. 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.

20. 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.

21. 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.

22. 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.

23. 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.

24. 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.

25. 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.

26. 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.

27. 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.

28. 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.

29. 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.

30. 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.

31. 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.

32. 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.

33. 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.

34. 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.

35. 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.

36. 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.

37. 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.

38. 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.

39. 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.

40. 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.

41. 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.

42. 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.

43. 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. – 2023. – P. 61-78.

44. Смірнов О.А. Козлов Я.О., Смірнова Т.В. «Дослідження застосування SIEM-систем для забезпечення кібербезпеки та захисту інформації». *II Міжнародна науково-практична Інтернет-конференція «Інновації та перспективні шляхи hозвитку інформаційних технологій (ІПШРІТ-2023)»* м.Черкаси 6 грудня 2023 року – Черкаси: ЧДТУ.– 2023. – С.251-252.

45. Козлов Я.О., Смірнова Т.В., Смірнов О.А. «Дослідження SIEM-систем для забезпечення кібербезпеки». *VII міжнародна науково-практична*

конференція "Інформаційна безпека та комп'ютерні технології" до 30-ти річчя кафедри кібербезпеки та програмного забезпечення, м. Кропивницький. 1 листопада 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 26.

46. Козлов Я.О., Козірова Н.Л., Смірнов О.А. «Дослідження структури та принципу роботи SIEM-системи». *VII міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології" до 30-ти річчя кафедри кібербезпеки та програмного забезпечення,* м. Кропивницький. 1 листопада 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 59.

47. Вінтенко Б.Ю., Смірнов О.А., Коваленко О.В., Смірнов С.А., Коваленко А.С. «Дослідження нормативних документів та галузевих стандартів розробки програмного забезпечення комп'ютерних систем управління АЕС, важливих для безпеки». *Системи управління, навігації та зв'язку*, 2023, вип. 2(72), С. 170-178.

48. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Смірнов С.А., Поліщук Л.І., «Дослідження стійкості до диференціального криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку,* 2022, № 3(69). С. 93-98.

49. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Поліщук Л.І., Смірнов С.А. «Дослідження статистичної стійкості та швидкісних характеристик запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Вісник Хмельницького національного університету. Серія: «Технічні науки»,* № 2 (307). С. 46-52. 2022.

50. Смірнов О.А., Смірнова Т.В., Константинова Л.В., Смірнов С.А., Якименко Н.М., «Дослідження стійкості до лінійного криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку*, 2022, № 1(67). С. 84-89.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку ієрархічних систем зберігання даних з використанням технології Tiered Storage.

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. № 132-02 від 01.04.2024 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення ієрархічних систем зберігання даних з використанням технології Tiered Storage.

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:

– ієрархічних систем зберігання даних з використанням технології Tiered Storage;

– 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 забезпеченням.

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 Delphi 10.4 Sydney.

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 – 90 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 23.05.2024 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 12.06.2024 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

_____ Усік П.С

Програмне забезпечення ієрархічних систем зберігання даних з використанням технології Tiered Storage

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 44

Літера: РП

Кропивницький – 2024 року

Файл Main.pas - Основна програма

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, ComCtrls, ExtCtrls, frFilePanel, StdCtrls, FileCtrl, ImgList,
  AppEvnts, Buttons, XPMan;

type
  TfmDAR = class(TForm)
    mmMenu: TMainMenu;
    miFile: TMenuItem;
    miExit: TMenuItem;
    miHelp: TMenuItem;
    miAbout: TMenuItem;
    frFilePanel: TfrFilePanel;
    pnTop: TPanel;
    sbStatus: TStatusBar;
    FileListBox1: TFileListBox;
    ImageList1: TImageList;
    miSplit1: TMenuItem;
    miAddToArchive: TMenuItem;
    miExtract: TMenuItem;
    miExtractTo: TMenuItem;
    miFileInformation: TMenuItem;
    bbAddToArchive: TBitBtn;
    bbExtractTo: TBitBtn;
    bbFileInformation: TBitBtn;
    lbPath: TLabel;
    lbItem: TLabel;
    SaveDialog1: TSaveDialog;
    ApplicationEvents1: TApplicationEvents;
    XPManifest1: TXPManifest;
    N1: TMenuItem;
    procedure miExitClick(Sender: TObject);
    procedure miAboutClick(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure miExtractToClick(Sender: TObject);
    procedure miAddToArchiveClick(Sender: TObject);
    procedure ApplicationEvents1Hint(Sender: TObject);
    procedure bbFileInformationClick(Sender: TObject);
    procedure frFilePanelbbRefreshClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    Procedure Compress;
    Procedure DeCompress;
  end;

var
  fmDAR: TfmDAR;

implementation

uses About, DeCompressor, fmExtractDir, fmProcess;//, frFilePanel;

Var
  FirstRun:Boolean;

{$R *.DFM}
```

```
Procedure TfmDAR.Compress;
Var
  NewFileName, OldFileName:String;
Begin
  NewFileName:=ChangeFileExt(lbItem.Caption, '.dar');
//  SaveDialog1.InitialDir:=lbPath.Caption;
  SaveDialog1.FileName:=NewFileName;

  If Not(SaveDialog1.Execute) Then Exit;
  NewFileName:=SaveDialog1.FileName;
  OldFileName:=lbPath.Caption+lbItem.Caption;
  If OldFileName=NewFileName Then
  Begin
    Application.MessageBox('Неможливо архівувати файл у себе', 'Помилка!',
MB_ICONERROR Or MB_OK);
    Exit;
  End;

  Enabled:=False;
  fmProcess.Compress(OldFileName, NewFileName);
  Enabled:=True;
End;

Procedure TfmDAR.DeCompress;
Var
  NewFileName, OldFileName:String;
Begin
  fmExtractDir.dlbxDirs.Directory:=lbPath.Caption;
  If fmExtractDir.ShowModal=mrCancel Then Exit;

  OldFileName:=lbPath.Caption+lbItem.Caption;
  NewFileName:=IncludeTrailingBackslash(fmExtractDir.dlbxDirs.Directory);

  Enabled:=False;
  fmProcess.DeCompress(OldFileName, NewFileName);
  Enabled:=True;
End;

Function _ShowProcess(OrigSize, OrigPos, Comp:Integer):Boolean;
Begin
  Result:=fmProcess.ShowProcess(OrigSize, OrigPos, Comp);
End;

procedure TfmDAR.miExitClick(Sender: TObject);
begin
  Close;
end;

procedure TfmDAR.miAboutClick(Sender: TObject);
begin
  fmAbout.ShowModal;
end;

procedure TfmDAR.FormActivate(Sender: TObject);
begin
  If FirstRun Then
  Begin
    frFilePanel.Init(FileListBox1, ImageList1, nil, lbPath, lbItem);
    FirstRun:=False;
  End;
end;

procedure TfmDAR.FormCreate(Sender: TObject);
begin
  FirstRun:=True;
  CompressProcessProc:=_ShowProcess;
end;
```

```
procedure TfmDAR.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  frFilePanel.Done;
end;
//розархівувати
procedure TfmDAR.miExtractToClick(Sender: TObject);
begin
  If UpperCase(ExtractFileExt(lbItem.Caption))<>'.DAR' Then Exit;
  DeCompress;
  frFilePanel.Refresh;
end;

//архівувати
procedure TfmDAR.miAddToArchiveClick(Sender: TObject);
begin
  Compress;
  frFilePanel.Refresh;
end;

procedure TfmDAR.ApplicationEvents1Hint(Sender: TObject);
begin
  sbStatus.Panels[0].Text:=Application.Hint;
end;
//довідка
procedure TfmDAR.bbFileInformationClick(Sender: TObject);
begin
 fmAbout.ShowModal;
end;
//обновити
procedure TfmDAR.frFilePanelbbRefreshClick(Sender: TObject);
begin
  frFilePanel.bbRefreshClick(Sender);

end;

end.
```

Файл Dictionary.pas - Створення словника

```
unit Dictionary;

interface

Const
  ConstBitsForDic=12;
  ConstWordsCount=1 Sh ConstBitsForDic;

Type
  TSingleWord=ANSIString;

  TDictionary=Object
    Words:Array[0.. ConstWordsCount-1] Of TSingleWord;
    Count:Integer;

    Constructor Init;
    Destructor Done;

    Procedure Clear;
    Function Add(SingleWord:TSingleWord):Integer;
    Function Find(SingleWord:TSingleWord):Integer;

    Function Compare(Word1, Word2:TSingleWord):Boolean;
  End;

Function GetWordLength(SingleWord:TSingleWord):Integer;
Function AddCharToWord(Var SingleWord:TSingleWord; Ch:Byte):Integer;
Procedure ClearWord(Var SingleWord:TSingleWord);

implementation

Constructor TDictionary.Init;
Begin
  Clear;
End;

Destructor TDictionary.Done;
Begin
  Clear;
End;

Procedure TDictionary.Clear;
Var
  i:Integer;
Begin
  For i:=0 To ConstWordsCount-1 Do
  Begin
    Words[i]:='';
  End;
  Count:=0;
End;

Function TDictionary.Add(SingleWord:TSingleWord):Integer;
Begin
  Result:=-1;
  If Count>=ConstWordsCount Then Exit;
  If SingleWord='' Then Exit;
  Result:=Find(SingleWord);
  If Result>=0 Then Exit;
  Words[Count]:=SingleWord;
  Result:=Count;
  Inc(Count);
End;

Function TDictionary.Find(SingleWord:TSingleWord):Integer;
```

```
Var
  i:Integer;
Begin
  Result:=-1;
  If Count<=0 Then Exit;
  If SingleWord='' Then Exit;
  For i:=0 To Count-1 Do
  Begin
    If SingleWord=Words[i] Then
    Begin
      Result:=i;
      Exit;
    End;
  End;
End;

Function TDictionary.Compare(Word1, Word2:TSingleWord):Boolean;
Begin
  Result:=(Word1=Word2);
End;

Function GetWordLength(SingleWord:TSingleWord):Integer;
Begin
  Result:=Length(SingleWord);
End;

Function AddCharToWord(Var SingleWord:TSingleWord; Ch:Byte):Integer;
Begin
  Result:=0;
  SingleWord:=SingleWord+Char(Ch);
End;

Procedure ClearWord(Var SingleWord:TSingleWord);
Begin
  SingleWord:='';
End;

end.
```

Файл Compressor.pas - Стиснення файлу

```
unit Compressor;

interface

Function CompressFile(SrcFile, DestFile:String):Integer;
Function DeCompressFile(SrcFile, DestDir:String):Integer;

implementation

Uses
  SysUtils, FileStreams, Dictionaries;

Var
  ReadStream:TFileReadStream;
  WriteStream:TFileWriteStream;
  Dic:TDictionary;
  NowBitsForDic:Byte;


Procedure WriteFileName(FileName:String; FSize:Integer);
Var
  i:Word;
Begin
  FileName:=ExtractFileName(FileName);
  WriteStream.Write(Length(FileName), 16);
  For i:=1 To Length(FileName) Do
    WriteStream.Write(Ord(FileName[i]), 8);
  WriteStream.Write(0, 8);

  i:=FSize And $FFFF;
  WriteStream.Write(i, 16);
  i:=FSize Sh 16;
  WriteStream.Write(i, 16);

  WriteStream.Write(0, 16);
End;

Procedure WriteSingleByte(Ch:Byte);
Begin
  WriteStream.WriteBit(0);
  WriteStream.Write(Ch, 8);
End;

Procedure WriteDoubleByte(Ch:Word; Cnt:Byte);
Begin
  WriteStream.WriteBit(1);
  WriteStream.Write(Ch, Cnt);
End;

Procedure WriteByDic(N:Integer);
Var
  Data:Word;
Begin
  If N<-256 Then Exit;
  Data:=Abs(N);
  If N<0 Then
  Begin
    Dec(Data);
    WriteSingleByte(Data);
  End
  Else Begin
    WriteDoubleByte(Data, NowBitsForDic);
  End;
End;
```

```
Function FindInDic(SingleWord:TSingleWord):Integer;
Begin
  If GetWordLength(SingleWord)=1 Then
  Begin
    Result:=-(Integer(SingleWord[1])+1);
    Exit;
  End;
  Result:=Dic.Find(SingleWord);
  If Result<0 Then Result:=-1024;
End;

Procedure ProcessNewByte(Var InpWord:TSingleWord; NewChar:Byte; Var
Out:Integer);
Var
  New:TSingleWord;
Begin
  New:=InpWord;
 {Якщо рядок уже занадто довга}
  If AddCharToWord(New, NewChar)<0 Then
  Begin
    Out:=FindInDic(New);  //запишемо цей рядок
    ClearWord(InpWord);
    AddCharToWord(InpWord, NewChar);  //Змінимо вхідне слово
    Exit;
  End;

  Out:=FindInDic(New);
  If Out>=-256 Then  //якщо є в словнику
  Begin  //те нічого не робимо
    Out:=-1024;
    InpWord:=New;  //Змінимо вхідне слово
  End
  Else Begin //Якщо немає в словнику, те...
    Dic.Add(New);  //додамо новий рядок у словник
    Out:=FindInDic(InpWord);  //а на вихід пошлемо попередній рядок
    ClearWord(InpWord);
    AddCharToWord(InpWord, NewChar);  //Змінимо вхідне слово
  End;
End;

Procedure ProcessCompression;
Var
  SrcWord:TSingleWord;
  NewByte:Byte;
  Out:Integer;
Begin
  NowBitsForDic:=ConstBitsForDic;

  If ReadStream.FileRead Then Exit;
  ClearWord(SrcWord);
  NewByte:=ReadStream.Read(8);
  AddCharToWord(SrcWord, NewByte);

  While Not(ReadStream.FileRead) Do
  Begin
    NewByte:=ReadStream.Read(8);
    ProcessNewByte(SrcWord, NewByte, Out);
    WriteByDic(Out);
  End;
  Out:=FindInDic(SrcWord);
  WriteByDic(Out);

  WriteStream.Write(0, 8-WriteStream.AdditionalBits+8);
End;

Function CompressFile(SrcFile, DestFile:String):Integer;
Var
  FSize:Integer;
Begin
```

```
  Result:=0;

  Dic.Clear;
  ReadStream.OpenStream(SrcFile);
  WriteStream.OpenStream(DestFile);

  FSize:=ReadStream.Size;

  WriteFileName(SrcFile, FSize);

  ProcessCompression;

  ReadStream.CloseStream;
  WriteStream.CloseStream;
End;

Function ReadFileName(Var FSize:Integer):String;
Var
  Len, i:Word;
  Tmp:Integer;
Begin
  Result:='';
  FSize:=0;

  Len:=ReadStream.Read(16);
  For i:=1 To Len Do
    Result:=Result+Char(ReadStream.Read(8));
  ReadStream.Read(8);

  FSize:=ReadStream.Read(16);

  Tmp:=ReadStream.Read(16);
  Tmp:=Tmp Sh 16;
  FSize:=Tmp+FSize;

  ReadStream.Read(16);
End;

Procedure SaveSingleWord(SingleWord:TSingleWord);
Var
  i:Word;
  Tmp:Byte;
Begin
  If GetWordLength(SingleWord)=0 Then Exit;
  For i:=1 To GetWordLength(SingleWord) Do
  Begin
    Tmp:=Byte(SingleWord[i]);
    WriteStream.Write(Tmp, 8);
  End;
End;

Procedure WriteByDicOrig(N:Integer);
Var
  Data:Word;
  SrcWord:TSingleWord;
Begin
  If N<-256 Then Exit;
  Data:=Abs(N);
  If N<0 Then
  Begin
    Dec(Data);
    WriteStream.Write(Data, 8);
  End
  Else Begin
    SrcWord:=Dic.Words[Data];
    SaveSingleWord(SrcWord);
  End;
End;
```

```
Procedure ProcessDeCompression(FSize:Integer);
Var
  SrcWord, SaveWord:TSingleWord;
  NewByte, BitFlag:Byte;
  Out, Temp:Integer;
  OrdinaryWord:Boolean;
  i:Integer;
Begin
  NowBitsForDic:=ConstBitsForDic;
  OrdinaryWord:=True;

  If FSize=0 Then Exit;

  ClearWord(SrcWord);

  While ((WriteStream.BytesWrote<FSize) And Not(ReadStream.FileRead)) Do
  Begin
    BitFlag:=ReadStream.ReadBit;
    If BitFlag=0 Then  //якщо це просто байт
    Begin
      NewByte:=ReadStream.Read(8);
      ProcessNewByte(SrcWord, NewByte, Out);
      WriteStream.Write(NewByte, 8);
    End
    Else Begin  //якщо це словникове слово
      Out:=ReadStream.Read(NowBitsForDic);
      If Out>=Dic.Count Then  //якщо потрібного слова немає в словнику
      Begin
        ProcessNewByte(SrcWord, Byte(SrcWord[1]), Temp);  //створюємо його з
випередженням
        OrdinaryWord:=False;
      End
      Else OrdinaryWord:=True;

      SaveWord:=Dic.Words[Out];
      SaveSingleWord(SaveWord);

      If OrdinaryWord Then  //якщо звичайне слово, те стандартний перебір
      Begin
        For i:=1 To GetWordLength(SaveWord) Do
        Begin
          NewByte:=Byte(SaveWord[i]);
          ProcessNewByte(SrcWord, NewByte, Out);
        End;
      End
      Else Begin  //інакше скорочений перебір
        For i:=2 To GetWordLength(SaveWord) Do
        Begin
          NewByte:=Byte(SaveWord[i]);
          ProcessNewByte(SrcWord, NewByte, Out);
        End;
      End;
    End;
  End;
End;

Function DeCompressFile(SrcFile, DestDir:String):Integer;
Var
  FSize:Integer;
  DestFile:String;
Begin
  Result:=0;

  Dic.Clear;
  ReadStream.OpenStream(SrcFile);

  DestFile:=DestDir+ReadFileName(FSize);
  WriteStream.OpenStream(DestFile);
```

```
  ProcessDeCompression(FSize);

  ReadStream.CloseStream;
  WriteStream.CloseStream;
End;


initialization
  Dic.Init;
  ReadStream:=TFileReadStream.Init;
  WriteStream:=TFileWriteStream.Init;

finalization
  Dic.Done;
  ReadStream.Done;
  WriteStream.Done;

end.
```

Файл DeCompressor.pas - Основна програма

```
unit DeCompressor;

interface

Const
  ConstFirstSignature:String[64]=
    'DAr compver0.9.0.0 do NOT change this data! blah-blah-blah!';

Type
  TCompressProcessProc=Function(OrigSize, OrigPos, Comp:Integer):Boolean;

Var
  CompressProcessProc:TCompressProcessProc;

Function CompressFile(SrcFile, DestFile:String):Integer;
Function DeCompressFile(SrcFile, DestDir:String):Integer;

implementation

Uses
  SysUtils, DFileStream, Dictionary, Main;

Var
  ReadStream:TFileReadStream;
  WriteStream:TFileWriteStream;
  Dic:TDictionary;
  NowBitsForDic:Byte;


Function CPP(OrigSize, OrigPos, Comp:Integer):Boolean;
Begin
//емуляція індикації процесу
  Result:=True;
End;

Procedure WriteSignature;
Var
  i:Byte;
Begin
  For i:=1 To Length(ConstFirstSignature) Do
    WriteStream.Write(Ord(ConstFirstSignature[i]), 8);
End;

Procedure WriteFileName(FileName:String; FSize:Integer);
Var
  i:Word;
Begin
  FileName:=ExtractFileName(FileName);
  WriteStream.Write(Length(FileName), 16);
  For i:=1 To Length(FileName) Do
    WriteStream.Write(Ord(FileName[i]), 8);
  WriteStream.Write(0, 8);

  i:=FSize And $FFFF;
  WriteStream.Write(i, 16);
  i:=FSize Sh 16;
  WriteStream.Write(i, 16);

  WriteStream.Write(0, 16);
End;

Procedure WriteSingleByte(Ch:Byte);
Begin
  WriteStream.WriteBit(0);
  WriteStream.Write(Ch, 8);
```

```
End;

Procedure WriteDoubleByte(Ch:Word; Cnt:Byte);
Begin
  WriteStream.WriteBit(1);
  WriteStream.Write(Ch, Cnt);
End;

Procedure WriteByDic(N:Integer);
Var
  Data:Word;
Begin
  If N<-256 Then Exit;
  Data:=Abs(N);
  If N<0 Then
  Begin
    Dec(Data);
    WriteSingleByte(Data);
  End
  Else Begin
    WriteDoubleByte(Data, NowBitsForDic);
  End;
End;

Function FindInDic(SingleWord:TSingleWord):Integer;
Begin
  If GetWordLength(SingleWord)=1 Then
  Begin
    Result:=-(Integer(SingleWord[1])+1);
    Exit;
  End;
  Result:=Dic.Find(SingleWord);
  If Result<0 Then Result:=-1024;
End;

Procedure ProcessNewByte(Var InpWord:TSingleWord; NewChar:Byte; Var
Out:Integer);
Var
  New:TSingleWord;
Begin
  New:=InpWord;
 {Якщо рядок уже занадто довга}
  If AddCharToWord(New, NewChar)<0 Then
  Begin
    Out:=FindInDic(New);  //запишемо цей рядок
    ClearWord(InpWord);
    AddCharToWord(InpWord, NewChar);  //Змінимо вхідне слово
    Exit;
  End;

  Out:=FindInDic(New);
  If Out>=-256 Then  //якщо є в словнику
  Begin  //те нічого не робимо
    Out:=-1024;
    InpWord:=New;  //Змінимо вхідне слово
  End
  Else Begin //Якщо немає в словнику, те...
    Dic.Add(New);  //додамо новий рядок у словник
    Out:=FindInDic(InpWord);  //а на вихід пошлемо попередній рядок
    ClearWord(InpWord);
    AddCharToWord(InpWord, NewChar);  //Змінимо вхідне слово
  End;
End;

Procedure ProcessCompression;
Var
  SrcWord:TSingleWord;
  NewByte:Byte;
  Out:Integer;
```

```
Begin
  NowBitsForDic:=ConstBitsForDic;

  If ReadStream.FileRead Then Exit;
  ClearWord(SrcWord);
  NewByte:=ReadStream.Read(8);
  AddCharToWord(SrcWord, NewByte);

  While Not(ReadStream.FileRead) Do
  Begin
    NewByte:=ReadStream.Read(8);
    ProcessNewByte(SrcWord, NewByte, Out);
    WriteByDic(Out);
   //Індикація
    If Not(CompressProcessProc(ReadStream.Size, ReadStream.BytesRead,
WriteStream.BytesWrote)) Then
    Begin
      Exit;
    End;
  End;
  Out:=FindInDic(SrcWord);
  WriteByDic(Out);

  WriteStream.Write(0, 8-WriteStream.AdditionalBits+8);
End;

Function CompressFile(SrcFile, DestFile:String):Integer;
Var
  FSize:Integer;
Begin
  Result:=0;

  Dic.Clear;
  ReadStream.OpenStream(SrcFile);
  WriteStream.OpenStream(DestFile);

  FSize:=ReadStream.Size;

  WriteSignature;
  WriteFileName(SrcFile, FSize);

  ProcessCompression;

  ReadStream.CloseStream;
  WriteStream.CloseStream;
End;

Function ReadSignature:Boolean;
Var
  i:Byte;
Begin
  Result:=False;
  For i:=1 To 64 Do
    If Ord(ConstFirstSignature[i])<>ReadStream.Read(8) Then Exit;
  Result:=True;
End;

Function ReadFileName(Var FSize:Integer):String;
Var
  Len, i:Word;
  Tmp:Integer;
Begin
  Result:='';
  FSize:=0;

  Len:=ReadStream.Read(16);
  For i:=1 To Len Do
    Result:=Result+Char(ReadStream.Read(8));
  ReadStream.Read(8);
```

```
  FSize:=ReadStream.Read(16);

  Tmp:=ReadStream.Read(16);
  Tmp:=Tmp Sh 16;
  FSize:=Tmp+FSize;

  ReadStream.Read(16);
End;

Procedure SaveSingleWord(SingleWord:TSingleWord);
Var
  i:Word;
  Tmp:Byte;
Begin
  If GetWordLength(SingleWord)=0 Then Exit;
  For i:=1 To GetWordLength(SingleWord) Do
  Begin
    Tmp:=Byte(SingleWord[i]);
    WriteStream.Write(Tmp, 8);
  End;
End;

Procedure WriteByDicOrig(N:Integer);
Var
  Data:Word;
  SrcWord:TSingleWord;
Begin
  If N<-256 Then Exit;
  Data:=Abs(N);
  If N<0 Then
  Begin
    Dec(Data);
    WriteStream.Write(Data, 8);
  End
  Else Begin
    SrcWord:=Dic.Words[Data];
    SaveSingleWord(SrcWord);
  End;
End;

Procedure ProcessDeCompression(FSize:Integer);
Var
  SrcWord, SaveWord:TSingleWord;
  NewByte, BitFlag:Byte;
  Out, Temp:Integer;
  OrdinaryWord:Boolean;
  i:Integer;
Begin
  NowBitsForDic:=ConstBitsForDic;
  OrdinaryWord:=True;

  If FSize=0 Then Exit;

  ClearWord(SrcWord);

  While ((WriteStream.BytesWrote<FSize) And Not(ReadStream.FileRead)) Do
  Begin
    BitFlag:=ReadStream.ReadBit;
    If BitFlag=0 Then  //якщо це просто байт
    Begin
      NewByte:=ReadStream.Read(8);
      ProcessNewByte(SrcWord, NewByte, Out);
      WriteStream.Write(NewByte, 8);
    End
    Else Begin  //якщо це словникове слово
      Out:=ReadStream.Read(NowBitsForDic);
      If Out>=Dic.Count Then  //якщо потрібного слова немає в словнику
      Begin
```

```
        ProcessNewByte(SrcWord, Byte(SrcWord[1]), Temp);  //створюємо його з
випередженням
        OrdinaryWord:=False;
      End
      Else OrdinaryWord:=True;

      SaveWord:=Dic.Words[Out];
      SaveSingleWord(SaveWord);

      If OrdinaryWord Then  //якщо звичайне слово, те стандартний перебір
      Begin
        For i:=1 To GetWordLength(SaveWord) Do
        Begin
          NewByte:=Byte(SaveWord[i]);
          ProcessNewByte(SrcWord, NewByte, Out);
        End;
      End
      Else Begin  //інакше скорочений перебір
        For i:=2 To GetWordLength(SaveWord) Do
        Begin
          NewByte:=Byte(SaveWord[i]);
          ProcessNewByte(SrcWord, NewByte, Out);
        End;
      End;
    End;
   //Індикація
    If Not(CompressProcessProc(FSize, WriteStream.BytesWrote,
ReadStream.BytesRead)) Then
    Begin
      Exit;
    End;
  End;
End;

Function DeCompressFile(SrcFile, DestDir:String):Integer;
Var
  FSize:Integer;
  DestFile:String;
Begin
  Result:=0;

  Dic.Clear;
  ReadStream.OpenStream(SrcFile);

  If Not(ReadSignature) Then
  Begin
    Result:=-1;
    Exit;
  End;
  DestFile:=DestDir+ReadFileName(FSize);
  WriteStream.OpenStream(DestFile);

  ProcessDeCompression(FSize);

  ReadStream.CloseStream;
  WriteStream.CloseStream;
End;


initialization
  Dic.Init;
  ReadStream:=TFileReadStream.Init;
  WriteStream:=TFileWriteStream.Init;
  CompressProcessProc:=CPP;

finalization
  Dic.Done;
  ReadStream.Done;
  WriteStream.Done;
```

```
end.
```

Файл fmProcess.pas - Візуалізація процесу архівування/розархівування

```
unit fmProcess;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, Gauges, ExtCtrls;

type
  TfmProcess = class(TForm)
    ggProcess: TGauge;
    ggRatio: TGauge;
    bbCabcel: TBitBtn;
    Label1: TLabel;
    Label2: TLabel;
    lbProcessInfo: TLabel;
    procedure bbCabcelClick(Sender: TObject);
  private
    { Private declarations }
    Continue:Boolean;
  public
    { Public declarations }
    Function ShowProcess(OrigSize, OrigPos, Comp:Integer):Boolean;
    Procedure Compress(OldFileName, NewFileName:String);
    Procedure DeCompress(OldFileName, NewFileName:String);
  end;

var
  fmProcess: TfmProcess;

implementation

uses DeCompressor;

{$R *.DFM}

Procedure TfmProcess.Compress(OldFileName, NewFileName:String);
Begin
  lbProcessInfo.Caption:='Архівування файлу "'+OldFileName+'"';
  Show;

  Continue:=True;
  CompressFile(OldFileName, NewFileName);
  Sleep(1000);
  Hide;
End;

Procedure TfmProcess.DeCompress(OldFileName, NewFileName:String);
Begin
  lbProcessInfo.Caption:='Розархівування файлу "'+OldFileName+'"';
  Show;

  Continue:=True;
  DeCompressFile(OldFileName, NewFileName);
  Sleep(1000);
  Hide;
End;

Function TfmProcess.ShowProcess(OrigSize, OrigPos, Comp:Integer):Boolean;
Begin
  ggProcess.MaxValue:=OrigSize;
  ggProcess.Progress:=OrigPos;

//  ggRatio.MaxValue:=OrigSize;
  ggRatio.MaxValue:=OrigPos;
```

```
  ggRatio.Progress:=Comp;

  Result:=Continue;
  Application.ProcessMessages;
End;

procedure TfmProcess.bbCabcelClick(Sender: TObject);
begin
  Continue:=False;
  Application.ProcessMessages;

  Application.MessageBox('Процес перервано користувачем', 'Відміна',
MB_ICONINFORMATION Or MB_OK);
  Hide;
end;

end.
```

Файл frFilePanel.pas - Інтерфейс

```
unit frFilePanel;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ComCtrls, StdCtrls, ExtCtrls, FileCtrl,
  Files, Buttons;

Type
  TColumnsSize=Array [0..4] Of Integer;
  TDeactivateProcedure=Procedure Of Object;
  TfrFilePanel = class(TFrame)
    pnDrives: TPanel;
    pnDriveInfo: TPanel;
    lbCurrentPath: TLabel;
    lvFiles: TListView;
    pnFilesInfo: TPanel;
    dcbxDrive: TDriveComboBox;
    btDirRoot: TButton;
    btDirUp: TButton;
    lbDriveInfo: TLabel;
    bbRefresh: TBitBtn;

    procedure lvFilesColumnClick(Sender: TObject; Column: TListColumn);
    procedure dcbxDriveChange(Sender: TObject);
    procedure lvFilesKeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
    procedure lvFilesDblClick(Sender: TObject);
    procedure btDirUpClick(Sender: TObject);
    procedure btDirRootClick(Sender: TObject);
    procedure lvFilesColumnRightClick(Sender: TObject; Column: TListColumn;
      Point: TPoint);
    procedure lvFilesEditing(Sender: TObject; Item: TListItem;
      var AllowEdit: Boolean);
    procedure lvFilesChange(Sender: TObject; Item: TListItem;
      Change: TItemChange);
    procedure lvFilesEnter(Sender: TObject);
    procedure lbCurrentPathClick(Sender: TObject);
    procedure bbRefreshClick(Sender: TObject);
  private
    { Private declarations }
    AllFiles:TFiles;

    SecondEdit:Boolean;
    Inited:Boolean;

    LastCaption, LastExt:String;

  public
    { Public declarations }
    flbxFiles:TFileListBox;
    CurrentFullPath:String;
    CurrentDrive:Char;
    CurrentPath:String;
    NowRoot:Boolean;

    SortColumn:Byte;
    SortAscending:Boolean;
    NowActive:Boolean;
    UseCopyToDir:String;
    OtherPanelDeactivate:TDeactivateProcedure;
    lbPathEx, lbItemEx:TLabel;
```

```
    Procedure Init(FilesListBox:TFileListBox; ImageList:TImageList;
Deactivation:TDeactivateProcedure; lbPath, lbItem:TLabel);
    Procedure Done;

    Procedure MakeOutLabels;
    Procedure Activate;
    Procedure Deactivate;
    Procedure CheckActive;

    Procedure Refresh;
    Procedure Sort;

    Procedure SetPath(Path:String);

    Procedure ShowItem(Item:TFileRecord);
    Procedure ShowFiles;

    Procedure GetItemByList(ListItem:TListItem; Var Item:TFileRecord);

    Procedure ShowInfo;
    Function ChangeCheck(ListItem:TListItem):Boolean;

    Procedure SetColumnsSize(ColumnsSize:TColumnsSize);
    Procedure GetColumnsSize(Var ColumnsSize:TColumnsSize);

    Procedure SelectLastItem;

    Function TryRename(ListItem:TListItem; NewName:String):Boolean;
    Function TryOneDelete(Item:TFileRecord):Integer;
    Function TryDelete:Boolean;
    Procedure TryCopyFile;
    Procedure TryMoveFile;

    Procedure EditFile;
    Procedure CreateFolder;

    Procedure SetDrive(Drive:Char);
    Procedure CheckCurrentPath;
  end;


implementation
Uses
  StrConsts, FilesEx, fmErrorDrive, fmNameQuery, fmAnyMessage,
  DeCompressor, Main;

{$R *.DFM}


Procedure TfrFilePanel.Init(FilesListBox:TFileListBox; ImageList:TImageList;
Deactivation:TDeactivateProcedure; lbPath, lbItem:TLabel);
Begin
  AllFiles.Init;

  flbxFiles:=FilesListBox;
  lvFiles.LargeImages:=ImageList;
  lvFiles.SmallImages:=ImageList;
  lvFiles.StateImages:=ImageList;
  SortColumn:=1;
  SortAscending:=True;

  SecondEdit:=False;
  Inited:=True;
  LastCaption:='';
  LastExt:='';
  OtherPanelDeactivate:=Deactivation;
  lbPathEx:=lbPath;
  lbItemEx:=lbItem;
{}
```

```
  Activate;
  SetDrive('C');
End;

Procedure TfrFilePanel.Done;
Begin
  Deactivate;
  AllFiles.Done;
End;

Procedure TfrFilePanel.MakeOutLabels;
Var
  Item:TFileRecord;
Begin
  If lbPathEx<>nil Then
  Begin
    lbPathEx.Caption:=CurrentFullPath;
    lbPathEx.Hint:=CurrentFullPath;
  End;

  If lbItemEx<>nil Then
  Begin
    GetItemByList(lvFiles.ItemFocused, Item);
    lbItemEx.Caption:=GetFullName(Item);
  End;
End;

Procedure TfrFilePanel.Activate;
Begin
  If @OtherPanelDeactivate<>nil Then OtherPanelDeactivate;
  NowActive:=True;
  UseCopyToDir:=ConstCopyToDir;
  lbCurrentPath.Color:=ConstLabelActiveColor;
  lvFiles.SetFocus;
  MakeOutLabels;
End;

Procedure TfrFilePanel.Deactivate;
Begin
  NowActive:=False;
  ConstCopyToDir:=CurrentFullPath;
  lbCurrentPath.Color:=ConstLabelNonActiveColor;
End;

Procedure TfrFilePanel.CheckActive;
Begin

End;

Procedure TfrFilePanel.Refresh;
Var
  i:Integer;
  Item:TFileRecord;
Begin
  CheckCurrentPath;

  AllFiles.Clear;

  flbxFiles.Directory:=CurrentFullPath;
  flbxFiles.Update;
  flbxFiles.FileType:=[ftReadOnly, ftHidden, ftSystem, ftArchive, ftDirectory,
ftNormal];
  If flbxFiles.Items.Count<=0 Then Exit;

  CurrentFullPath:=IncludeTrailingBackslash(flbxFiles.Directory);
  CurrentDrive:=ExtractFileDrive(CurrentFullPath)[1];

  NowRoot:=IsRoot(CurrentFullPath);
```

```
  For i:=0 To flbxFiles.Items.Count-1 Do
  Begin
    GetItemByFileName(flbxFiles.Items[i], Item);
    If IsDirectory(Item) Then
    Begin
      If ((Item.Name<>'.') And (Item.Name<>'..') And (Item.Name<>'')) Then
        AllFiles.Add(Item);
    End
    Else Begin
      AllFiles.Add(Item);
    End;
  End;
  Sort;
  ShowFiles;

  SelectLastItem;
  ShowInfo;

 {$ I-I-}
  If IOResult<>0 Then MessageBeep(48);
 {$I+}
End;

Procedure TfrFilePanel.Sort;
Var
  i:Integer;
  ColCapt:String;
Begin
  For i:=0 To lvFiles.Columns.Count-1 Do
  Begin
    ColCapt:=lvFiles.Column[i].Caption;
    Delete(ColCapt, Length(ColCapt), 1);
    lvFiles.Column[i].Caption:=ColCapt+ConstNoSort;
  End;
  ColCapt:=lvFiles.Column[SortColumn].Caption;
  Delete(ColCapt, Length(ColCapt), 1);
  If SortAscending Then
    lvFiles.Column[SortColumn].Caption:=ColCapt+ConstSortAscending
  Else
    lvFiles.Column[SortColumn].Caption:=ColCapt+ConstSortDescending;

  Case SortColumn Of
    0: AllFiles.SortByName(SortAscending);
    1: AllFiles.SortByExt(SortAscending);
    2: AllFiles.SortBySize(SortAscending);
    3: AllFiles.SortByDateTime(SortAscending);
    4: AllFiles.SortByAttr(SortAscending);
  Else
    AllFiles.SortByName(SortAscending);
  End;
End;

procedure TfrFilePanel.lvFilesColumnClick(Sender: TObject;
  Column: TListColumn);
begin
  If lvFiles.ItemFocused<>nil Then
  Begin
    LastCaption:=lvFiles.ItemFocused.Caption;
    LastExt:=lvFiles.ItemFocused.SubItems[0];
  End
  Else Begin
    LastCaption:='';
    LastExt:='';
  End;

  If Column.Index=SortColumn Then
    SortAscending:=Not(SortAscending)
  Else
    SortAscending:=True;
```

```
  SortColumn:=Column.Index;
  Sort;
  ShowFiles;

  SelectLastItem;
end;

Procedure TfrFilePanel.SetPath(Path:String);
Var
  TmpStr:String;
Begin
  TmpStr:=ExcludeTrailingBackslash(Path);
  TmpStr:=ExtractFileName(TmpStr);

  If TmpStr='..' Then
  Begin
    LastCaption:=ExcludeTrailingBackslash(CurrentFullPath);

LastCaption:=ConstDirLeftBracket+ExtractFileName(LastCaption)+ConstDirRightBrack
et;
    LastExt:='';
  End;

  CurrentFullPath:=Path;

  Refresh;
End;

Procedure TfrFilePanel.ShowItem(Item:TFileRecord);
Var
  FormattedItem:TFileFormattedRecord;
  ListItem:TListItem;
Begin

    GetFormattedItem(Item, FormattedItem);
    ListItem:=lvFiles.Items.Add;
    With ListItem Do
    Begin
      ImageIndex:=GetItemImageIndex(Item);
      If Item.Checked Then
        StateIndex:=0
      Else
        StateIndex:=-1;
      If IsDirectory(Item) Then
        Caption:=ConstDirLeftBracket+FormattedItem.Name+ConstDirRightBracket
      Else
        Caption:=FormattedItem.Name;
      SubItems.Add(FormattedItem.Ext);
      SubItems.Add(FormattedItem.Size);
      SubItems.Add(FormattedItem.DateTime);
      SubItems.Add(FormattedItem.Attr);
    End;

End;

Procedure TfrFilePanel.ShowFiles;
Var
  i:Integer;
  Item:TFileRecord;
Begin
  lvFiles.Items.Clear;

  If NowRoot Then
    lvFiles.AllocBy:=AllFiles.ItemsCount
  Else
    lvFiles.AllocBy:=AllFiles.ItemsCount+1;

  If Not(NowRoot) Then ShowItem(DirUpItem);
```

```
  For i:=0 To AllFiles.ItemsCount-1 Do
  Begin
    AllFiles.GetItem(i, Item);

    ShowItem(Item);

  End;

End;

Procedure TfrFilePanel.GetItemByList(ListItem:TListItem; Var Item:TFileRecord);
Var
  i:Integer;
Begin
  i:=lvFiles.Items.IndexOf(ListItem);
  If Not(NowRoot) Then Dec(i);
  If i<0 Then
    Item:=DirUpItem
  Else
    AllFiles.GetItem(i, Item);
End;

Procedure TfrFilePanel.ShowInfo;
Var
  TotalBytes, TotalFree:Int64;
  TotalDirs, TotalFiles, CheckedDirs, CheckedFiles:Integer;
  TotalSize, CheckedSize:Int64;
  TmpStr:String;
Begin
  GetDiskSize(CurrentDrive, TotalBytes, TotalFree);
  lbCurrentPath.Caption:=CurrentFullPath;
  lbCurrentPath.Hint:=CurrentFullPath;
  TmpStr:=GetCompactSize(TotalFree)+' з '+GetCompactSize(TotalBytes)+' вільно';
  lbDriveInfo.Caption:=TmpStr;
  lbDriveInfo.Hint:=TmpStr;

  AllFiles.GetInfo(TotalDirs, TotalFiles, CheckedDirs, CheckedFiles, TotalSize,
CheckedSize);
  TmpStr:=' '+GetCompactSize(CheckedSize)+' / '+GetCompactSize(TotalSize)+' в '+
    GetFormattedSize(CheckedFiles, 1, True)+' / '+GetFormattedSize(TotalFiles,
1, True)+' файлі(ах) і '+
    GetFormattedSize(CheckedDirs, 1, True)+' / '+GetFormattedSize(TotalDirs, 1,
True)+' папці(ках)';
  pnFilesInfo.Caption:=TmpStr;
  pnFilesInfo.Hint:=TmpStr;
End;

Procedure TfrFilePanel.SetColumnsSize(ColumnsSize:TColumnsSize);
Var
  i:Integer;
Begin
  For i:=0 To lvFiles.Columns.Count-1 Do
    lvFiles.Columns.Items[i].Width:=ColumnsSize[i];
End;

Procedure TfrFilePanel.GetColumnsSize(Var ColumnsSize:TColumnsSize);
Var
  i:Integer;
Begin
  For i:=0 To lvFiles.Columns.Count-1 Do
    ColumnsSize[i]:=lvFiles.Columns.Items[i].Width;
End;

procedure TfrFilePanel.dcbxDriveChange(Sender: TObject);
begin
  If Not(Inited) Then Exit;
  SetDrive(dcbxDrive.Drive);
  Refresh;
  Activate;
```

```
end;

procedure TfrFilePanel.lvFilesKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
Var
  TmpStr:String;
  TmpBool:Boolean;
begin
  Case Key Of
    VK_Return:Begin
        lvFilesDblClick(Self);
      End;
    VK_Left:Begin
        lvFiles.ItemFocused:=lvFiles.Items.Item[0];
        lvFiles.Selected:=lvFiles.Items.Item[0];
      End;
    VK_Right:Begin
        lvFiles.ItemFocused:=lvFiles.Items.Item[lvFiles.Items.Count-1];
        lvFiles.Selected:=lvFiles.Items.Item[lvFiles.Items.Count-1];
      End;
    VK_Back:Begin
        If ssCtrl In Shift Then
          btDirRootClick(Self)
        Else
          btDirUpClick(Self);
      End;
    VK_Delete:Begin
        TryDelete;
      End;
    VK_F8:Begin
        TryDelete;
      End;
    VK_F2:Begin
        lvFilesEditing(Self, lvFiles.ItemFocused, TmpBool);
      End;
    VK_F3:Begin
        EditFile;
      End;
    VK_F4:Begin
        If ssShift In Shift Then
        Begin
          TmpStr:=ConstLastRequest;
          If Not(GetNameQuery('Редагувати файл:', TmpStr)) Then Exit;
          ExecuteOneFile(CurrentFullPath, ConstNotepadFile, TmpStr);
          Exit;
        End;
        EditFile;
      End;
    VK_F5:Begin
        TryCopyFile;
      End;
    VK_F6:Begin
        TryMoveFile;
      End;
    VK_F7:Begin
        CreateFolder;
      End;
  End;
end;

Function TfrFilePanel.ChangeCheck(ListItem:TListItem):Boolean;
Var
  Item:TFileRecord;
  ID:Integer;
Begin
  Result:=False;
  GetItemByList(ListItem, Item);
  If Item.Name=ConstDirUp Then Exit;
  ID:=Item.ID;
```

```
  AllFiles.Items[ID].Checked:=Not(AllFiles.Items[ID].Checked);
  If AllFiles.Items[ID].Checked Then ListItem.StateIndex:=ConstCheckedImageIndex
Else ListItem.StateIndex:=ConstUnCheckedImageIndex;
  Result:=AllFiles.Items[ID].Checked;
  ShowInfo;
End;

Procedure TfrFilePanel.SelectLastItem;
Var
  ListItem:TListItem;
  StartIndex:Integer;
Begin
  StartIndex:=0;
  Repeat
    ListItem:=lvFiles.FindCaption(StartIndex, LastCaption, False, False, False);
    If ListItem=nil Then
    Begin
      lvFiles.ItemFocused:=lvFiles.Items.Item[0];
      lvFiles.Selected:=lvFiles.Items.Item[0];
      LastCaption:='';
      LastExt:='';
      Exit;
    End;
    If ListItem.SubItems[0]=LastExt Then
    Begin
      lvFiles.ItemFocused:=ListItem;
      lvFiles.Selected:=ListItem;
      LastCaption:='';
      LastExt:='';
      Exit;
    End;
    StartIndex:=ListItem.Index;
  Until False;
End;

Function TfrFilePanel.TryRename(ListItem:TListItem; NewName:String):Boolean;
Var
  Item:TFileRecord;
  OldName, Name, Ext:String;
  Tmp:Integer;
  TmpStr:String;
Begin
  Result:=False;
  Name:=ExtractFileName(NewName);
  Ext:=ExtractFileExt(NewName);
//  NewName:=CurrentFullPath+Name;
  If Ext<>'' Then
  Begin
    Delete(Name, Length(Name)-Length(Ext)+1, Length(Ext));
    Delete(Ext, 1, 1);
  End;
  If Name='' Then Exit;

  GetItemByList(ListItem, Item);
  OldName:=CurrentFullPath+GetFullName(Item);
  Tmp:=RenameOneFile(OldName, NewName);
  If Tmp<0 Then
  Begin
    TmpStr:=GetFileError(Tmp)+#0;
    Application.MessageBox(@TmpStr[1], 'Помилка!', MB_ICONERROR Or MB_OK);
    Exit;
  End;

  If IsDirectory(Item) Then
    LastCaption:=ConstDirLeftBracket+Name+ConstDirRightBracket
  Else
    LastCaption:=Name;
  LastExt:=Ext;
```

```
  Result:=True;
End;

Function TfrFilePanel.TryOneDelete(Item:TFileRecord):Integer;
Begin
  If Item.Name=ConstDirUp Then
  Begin
    Result:=F_ER_ERROR;
    Exit;
  End;
  If IsDirectory(Item) Then
    Result:=DeleteOneDir(CurrentFullPath+Item.Name)
  Else
    Result:=DeleteOneFile(CurrentFullPath+GetFullName(Item));
End;

Function TfrFilePanel.TryDelete:Boolean;
Var
  ListItem:TListItem;
  Item:TFileRecord;
  Tmp:Integer;
  TmpStr:String;
Begin
  Result:=False;
  ListItem:=lvFiles.ItemFocused;
  GetItemByList(ListItem, Item);
  If Item.Name=ConstDirUp Then Exit;

  TmpStr:='Ви дійсно хочете видалити "'+GetFullName(Item)+'"?'+#0;
  If Application.MessageBox(@TmpStr[1], 'Попередження', MB_ICONQUESTION Or
MB_YESNO)=IDNO Then Exit;

  Tmp:=TryOneDelete(Item);
  If Tmp=F_ER_SUCCESS Then
  Begin
    Result:=True;
    Refresh;
    Exit;
  End;
  TmpStr:=GetFileError(Tmp)+#0;
  Application.MessageBox(@TmpStr[1], 'Помилка!', MB_ICONERROR Or MB_OK);
  Result:=False;
End;

Procedure TfrFilePanel.TryCopyFile;
Var
  Item:TFileRecord;
  FileName, TmpStr:String;
  Tmp:Integer;
Begin
  GetItemByList(lvFiles.ItemFocused, Item);
  If IsDirectory(Item) Then
  Begin
    Application.MessageBox('Неможливо копіювати папку', 'Помилка!', MB_ICONERROR
Or MB_OK);
    Exit;
  End;
  FileName:=UseCopyToDir+GetFullName(Item);
  If Not(GetNameQuery('Скопіювати файл "'+GetFullName(Item)+'" в:', FileName))
Then Exit;
  If ExtractFileName(FileName)=FileName Then
    FileName:=CurrentFullPath+FileName;
  ShowAnyMessage('Копіювання...', 'Копіюється файл
"'+CurrentFullPath+GetFullName(Item)+'" в "'+FileName+'"');
  Tmp:=CopyOneFile(CurrentFullPath+GetFullName(Item), FileName, True);
  HideAnyMessage;
  If Tmp=F_ER_SUCCESS Then
  Begin
    MessageBeep(0);
```

```
    LastCaption:=Item.Name;
    LastExt:=Item.Ext;
    Refresh;
    Exit;
  End;
  TmpStr:=GetFileError(Tmp)+#0;
  Application.MessageBox(@TmpStr[1], 'Помилка!', MB_ICONERROR Or MB_OK);
End;

Procedure TfrFilePanel.TryMoveFile;
Var
  Item:TFileRecord;
  NewName:String;
Begin
  GetItemByList(lvFiles.ItemFocused, Item);
  If Item.Name=DirUpItem.Name Then Exit;
  NewName:=UseCopyToDir+GetFullName(Item);

  If GetNameQuery('Перемістити "'+GetFullName(Item)+'" в:', NewName) Then
  Begin
    If TryRename(lvFiles.ItemFocused, NewName) Then Refresh;
  End;
End;

Procedure TfrFilePanel.EditFile;
Var
  ListItem:TListItem;
  Item:TFileRecord;
Begin
  ListItem:=lvFiles.ItemFocused;
  GetItemByList(ListItem, Item);
  If IsDirectory(Item) Then Exit;

  ExecuteOneFile(CurrentFullPath, ConstNotepadFile, GetFullName(Item));
End;

Procedure TfrFilePanel.CreateFolder;
Var
  FolderName:String;
  Tmp:Integer;
  TmpStr:String;
Begin
  FolderName:=ConstLastRequest;
  If Not(GetNameQuery('Створити папку:', FolderName)) Then Exit;
  If FolderName='' Then Exit;
  Tmp:=CreateOneFolder(CurrentFullPath+FolderName);
  If Tmp=F_ER_SUCCESS  Then
  Begin
    LastCaption:=ConstDirLeftBracket+FolderName+ConstDirRightBracket;
    LastExt:='';
    Refresh;
    Exit;
  End;
  TmpStr:=GetFileError(Tmp)+#0;
  Application.MessageBox(@TmpStr[1], 'Помилка!', MB_ICONERROR Or MB_OK);
End;

Procedure TfrFilePanel.SetDrive(Drive:Char);
Var
  Dir:String;
Begin
  Repeat
    Drive:=UpCase(Drive);
   {$ I-I-}
    GetDir(Ord(Drive)-64, Dir);
    If Drive=Dir[1] Then
      ChDir(Dir)
    Else Begin
      Dir:=Drive+':\';
```

```
      ChDir(Dir);
    End;
    If IOResult<>0 Then
    Begin
      Dir:=Drive+':\';
      ChDir(Dir);
    End;
   {$I+}
    If IOResult=0 Then
    Begin
      dcbxDrive.Drive:=Drive;
      CurrentDrive:=Drive;
      CurrentFullPath:=Dir;
      Exit;
    End;
    ChooseNewDrive(Drive);
  Until False;
End;

Procedure TfrFilePanel.CheckCurrentPath;
Var
  Dir:String;
Begin
 {$ I-I-}
  ChDir(CurrentFullPath);
 {$I+}
  If IOResult<>0 Then
  Begin
    SetDrive(ExtractFileDrive(CurrentFullPath)[1]);
  End;
  Dir:=CurrentFullPath+#0;
  SetCurrentDirectory(@Dir[1]);
End;


procedure TfrFilePanel.lvFilesDblClick(Sender: TObject);
Var
  ListItem:TListItem;
  Item:TFileRecord;
  ErrorCode:Integer;
  ErrorString:String;
begin
  ListItem:=lvFiles.ItemFocused;
  If ListItem=nil Then Exit;

  GetItemByList(ListItem, Item);

  If IsDirectory(Item) Then
  Begin
    SetPath(CurrentFullPath+Item.Name);
    Exit;
  End;

  If UpperCase(Item.Ext)=ConstArchiveExt Then
  Begin
    fmDAR.DeCompress;
    Refresh;
    Exit;
  End;

  fmDAR.Compress;
  Refresh;
end;

procedure TfrFilePanel.btDirUpClick(Sender: TObject);
begin
  Activate;
  SetPath(CurrentFullPath+'..');
end;
```

```
procedure TfrFilePanel.btDirRootClick(Sender: TObject);
begin
  Activate;
  SetPath(IncludeTrailingBackslash(ExtractFileDrive(CurrentFullPath)));
end;

procedure TfrFilePanel.lvFilesColumnRightClick(Sender: TObject;
  Column: TListColumn; Point: TPoint);
begin
  Activate;
  Column.Width:=-1;
end;

procedure TfrFilePanel.lvFilesEditing(Sender: TObject; Item: TListItem;
  var AllowEdit: Boolean);
Var
  FileItem:TFileRecord;
  NewName:String;
begin
  AllowEdit:=False;
  GetItemByList(Item, FileItem);
  If FileItem.Name=DirUpItem.Name Then Exit;

  NewName:=GetFullName(FileItem);
  If GetNameQuery('Перейменувати "'+NewName+'" в:', NewName) Then
  Begin
    If TryRename(Item, NewName) Then Refresh;
  End;

  AllowEdit:=False;
end;

procedure TfrFilePanel.lvFilesChange(Sender: TObject; Item: TListItem;
  Change: TItemChange);
Begin
  MakeOutLabels;
end;

procedure TfrFilePanel.lvFilesEnter(Sender: TObject);
begin
  Activate;
end;

procedure TfrFilePanel.lbCurrentPathClick(Sender: TObject);
begin
  Activate;
end;

procedure TfrFilePanel.bbRefreshClick(Sender: TObject);
begin
  Refresh;
  Activate;
end;

end.
```

Файл Files.pas - Робота з файлами

```
  unit Files;

interface

Uses
  classes, SysUtils, FileCtrl,
  StrConsts;

Const
  MaxFilesCount=10240;
  MaxFileTypes=36;

Type
  TImagesIndex=Array [0.. MaxFileTypes-1, 0..1] Of String;

  TRecordID=Object
    ID:Integer;
  End;
  TFileRecord=Record
    ID:Integer;
    Name:String;
    Ext:String;
    Size:Integer;
    DateTime:TDateTime;
    Attr:Integer;
    Checked:Boolean;
    Tag:Integer;
  End;
  TFileFormattedRecord=Record
    Name,
    Ext,
    Size,
    DateTime,
    Attr:String;
    Checked:Boolean;
    Tag:Integer;
  End;

  TFiles=Object
  Protected
    slDirs, slFiles:TStringList;
    DirsID, FilesID:Array[0.. MaxFilesCount-1] Of TRecordID;

    Procedure FinishSort(Ascending, DirAscending:Boolean);
  Public
    ItemsID:Array[0.. MaxFilesCount-1] Of Integer;
    Items:Array[0.. MaxFilesCount-1] Of TFileRecord;
    ItemsCount:Integer;
    Tag:Integer;

    Constructor Init;
    Destructor Done;

    Function Add(Item:TFileRecord):Boolean;
    Procedure Clear;
    Procedure GetItem(ItemNo:Integer; Var Item:TFileRecord);

    Procedure SortByName(Ascending:Boolean);
    Procedure SortByExt(Ascending:Boolean);
    Procedure SortBySize(Ascending:Boolean);
    Procedure SortByDateTime(Ascending:Boolean);
    Procedure SortByAttr(Ascending:Boolean);
    Procedure GetInfo(Var TotalDirs, TotalFiles, CheckedDirs,
CheckedFiles:Integer; Var TotalSize, CheckedSize:Int64);
  End;
```

```
Const
  DirUpItem:TFileRecord=(
    ID:0;
    Name:ConstDirUp;
    Ext:'';
    Size:0;
    DateTime:0;
    Attr:faDirectory;
    Checked:False;
    Tag:0);

  Images:TImagesIndex=(
    ('','19'),
    ('EXE','14'),
    ('BAT','14'),
    ('COM','14'),
    ('LNK','19'),
    ('PIF','17'),
    ('TXT','20'),
    ('HTM','21'),
    ('HTML','21'),
    ('DOC','22'),
    ('DOT','22'),
    ('XLS','23'),
    ('RAR','26'),
    ('ZIP','27'),
    ('ARJ','27'),
    ('PAS','28'),
    ('DPR','28'),
    ('DFM','28'),
    ('DCU','28'),
    ('MP3','30'),
    ('M3U','31'),
    ('WAV','30'),
    ('MID','30'),
    ('OGG','30'),
    ('AVI','32'),
    ('MPE','32'),
    ('MPG','32'),
    ('MPEG','32'),
    ('BMP','34'),
    ('GIF','33'),
    ('JPG','33'),
    ('PCX','33'),
    ('ICO','33'),
    ('INI','20'),
    ('REG','35'),
    ('DAR','36'));

Function IsDirectory(Item:TFileRecord):Boolean;
Function IsRoot(Path:String):Boolean;

Function GetFormattedName(Name:String):String;
Function GetFormattedExt(Ext:String):String;
Function GetFormattedSize(Size:Integer; SizeWidth:Byte;
UseSeparators:Boolean):String;
Function GetFormattedDateTime(DateTime:TDateTime):String;
Function GetFormattedAttr(Attr:Integer):String;
Procedure GetFormattedItem(Item:TFileRecord; Var
FormattedItem:TFileFormattedRecord);

Procedure GetItemByFileName(FileName:String; Var Item:TFileRecord);

Function GetItemImageIndex(Item:TFileRecord):Integer;

Function GetCompactSize(Size:Int64):String;

Function GetFullName(Item:TFileRecord):String;
```

```
implementation


Constructor TFiles.Init;
Begin
  slDirs:=TStringList.Create;
  slFiles:=TStringList.Create;
  ItemsCount:=0;
  Tag:=0;
End;

Destructor TFiles.Done;
Begin
  slDirs.Free;
  slFiles.Free;
End;

Function TFiles.Add(Item:TFileRecord):Boolean;
Begin
  Result:=False;
  If ItemsCount>=MaxFilesCount Then Exit;
  Result:=True;
  Item.ID:=ItemsCount;
  Items[ItemsCount]:=Item;
  ItemsID[ItemsCount]:=ItemsCount;
  Inc(ItemsCount);
End;

Procedure TFiles.Clear;
Begin
  slDirs.Clear;
  slFiles.Clear;
  ItemsCount:=0;
End;

Procedure TFiles.GetItem(ItemNo:Integer; Var Item:TFileRecord);
Begin
  If ((ItemNo<0) Or (ItemNo>=ItemsCount)) Then Exit;
  Item:=Items[ItemsID[ItemNo]];
End;

Procedure TFiles.FinishSort(Ascending, DirAscending:Boolean);
Var
  i:Integer;
Begin
  slDirs.Sort;
  slFiles.Sort;

  If slDirs.Count>0 Then
    For i:=0 To slDirs.Count-1 Do
      If DirAscending Then
        ItemsID[i]:=TRecordID(slDirs.Objects[i]).ID
      Else
        ItemsID[i]:=TRecordID(slDirs.Objects[slDirs.Count-i-1]).ID;
  If slFiles.Count>0 Then
    For i:=slDirs.Count To slFiles.Count-1+slDirs.Count Do
      If Ascending Then
        ItemsID[i]:=TRecordID(slFiles.Objects[ i-slDirs.Count]).ID
      Else
        ItemsID[i]:=TRecordID(slFiles.Objects[slFiles.Count+slDirs.Count-i-
1]).ID;
End;

Procedure TFiles.SortByName(Ascending:Boolean);
Var
  Item:TFileRecord;
  i:Integer;
Begin
  If ItemsCount<=0 Then Exit;
```

```
  slDirs.Clear;
  slFiles.Clear;
  For i:=0 To ItemsCount-1 Do
  Begin
    Item:=Items[i];
    If IsDirectory(Item) Then
    Begin
      slDirs.Add(GetFormattedName(Item.Name));
      DirsID[slDirs.Count-1].ID:=i;
      slDirs.Objects[slDirs.Count-1]:=TObject(DirsID[slDirs.Count-1]);
    End
    Else Begin
      slFiles.Add(GetFormattedName(Item.Name));
      FilesID[slFiles.Count-1].ID:=i;
      slFiles.Objects[slFiles.Count-1]:=TObject(FilesID[slFiles.Count-1]);
    End;
  End;
  FinishSort(Ascending, Ascending);
End;

Procedure TFiles.SortByExt(Ascending:Boolean);
Var
  Item:TFileRecord;
  i:Integer;
Begin
  If ItemsCount<=0 Then Exit;
  slDirs.Clear;
  slFiles.Clear;
  For i:=0 To ItemsCount-1 Do
  Begin
    Item:=Items[i];
    If IsDirectory(Item) Then
    Begin
      slDirs.Add(GetFormattedName(Item.Name));
      DirsID[slDirs.Count-1].ID:=i;
      slDirs.Objects[slDirs.Count-1]:=TObject(DirsID[slDirs.Count-1]);
    End
    Else Begin
      slFiles.Add(GetFormattedExt(Item.Ext));
      FilesID[slFiles.Count-1].ID:=i;
      slFiles.Objects[slFiles.Count-1]:=TObject(FilesID[slFiles.Count-1]);
    End;
  End;
  FinishSort(Ascending, True);
End;

Procedure TFiles.SortBySize(Ascending:Boolean);
Var
  Item:TFileRecord;
  i:Integer;
  TmpStr:String;
Begin
  If ItemsCount<=0 Then Exit;
  slDirs.Clear;
  slFiles.Clear;
  For i:=0 To ItemsCount-1 Do
  Begin
    Item:=Items[i];
    If IsDirectory(Item) Then
    Begin
      slDirs.Add(GetFormattedName(Item.Name));
      DirsID[slDirs.Count-1].ID:=i;
      slDirs.Objects[slDirs.Count-1]:=TObject(DirsID[slDirs.Count-1]);
    End
    Else Begin
      TmpStr:=GetFormattedSize(Item.Size, 10, False);
      slFiles.Add(TmpStr);
      FilesID[slFiles.Count-1].ID:=i;
      slFiles.Objects[slFiles.Count-1]:=TObject(FilesID[slFiles.Count-1]);
```

```
    End;
  End;
  FinishSort(Ascending, True);
End;

Procedure TFiles.SortByDateTime(Ascending:Boolean);
Var
  Item:TFileRecord;
  i:Integer;
  TmpStr:String;
Begin
  If ItemsCount<=0 Then Exit;
  slDirs.Clear;
  slFiles.Clear;
  For i:=0 To ItemsCount-1 Do
  Begin
    Item:=Items[i];
    If IsDirectory(Item) Then
    Begin
      slDirs.Add(GetFormattedName(Item.Name));
      DirsID[slDirs.Count-1].ID:=i;
      slDirs.Objects[slDirs.Count-1]:=TObject(DirsID[slDirs.Count-1]);
    End
    Else Begin
      Str(Item.DateTime:16:10, TmpStr);
      slFiles.Add(TmpStr);
      FilesID[slFiles.Count-1].ID:=i;
      slFiles.Objects[slFiles.Count-1]:=TObject(FilesID[slFiles.Count-1]);
    End;
  End;
  FinishSort(Ascending, True);
End;

Procedure TFiles.SortByAttr(Ascending:Boolean);
Var
  Item:TFileRecord;
  i:Integer;
Begin
  If ItemsCount<=0 Then Exit;
  slDirs.Clear;
  slFiles.Clear;
  For i:=0 To ItemsCount-1 Do
  Begin
    Item:=Items[i];
    If IsDirectory(Item) Then
    Begin
      slDirs.Add(GetFormattedName(Item.Name));
      DirsID[slDirs.Count-1].ID:=i;
      slDirs.Objects[slDirs.Count-1]:=TObject(DirsID[slDirs.Count-1]);
    End
    Else Begin
      slFiles.Add(GetFormattedAttr(Item.Attr));
      FilesID[slFiles.Count-1].ID:=i;
      slFiles.Objects[slFiles.Count-1]:=TObject(FilesID[slFiles.Count-1]);
    End;
  End;
  FinishSort(Ascending, True);
End;

Procedure TFiles.GetInfo(Var TotalDirs, TotalFiles, CheckedDirs,
CheckedFiles:Integer; Var TotalSize, CheckedSize:Int64);
Var
  i:Integer;
Begin
  TotalDirs:=0;
  TotalFiles:=0;
  CheckedDirs:=0;
  CheckedFiles:=0;
  TotalSize:=0;
```

```
  CheckedSize:=0;

  If ItemsCount<=0 Then Exit;
  For i:=0 To ItemsCount-1 Do
  Begin
    If IsDirectory(Items[i]) Then
    Begin
      Inc(TotalDirs);
      If Items[i].Checked Then Inc(CheckedDirs);
    End
    Else Begin
      Inc(TotalFiles);
      If Items[i].Checked Then Inc(CheckedFiles);
    End;
    If Items[i].Checked Then CheckedSize:=CheckedSize+Items[i].Size;
    TotalSize:=TotalSize+Items[i].Size;
  End;
End;


Function IsDirectory(Item:TFileRecord):Boolean;
Begin
  If ((Item.Attr And faDirectory)>0) Then Result:=True Else Result:=False;
End;

Function IsRoot(Path:String):Boolean;
Begin
  Result:=(Path=IncludeTrailingBackslash(ExtractFileDrive(Path)));
End;

Function GetFormattedName(Name:String):String;
Begin
  Result:=Name;
End;

Function GetFormattedExt(Ext:String):String;
Begin
  Result:=Ext;
End;

Function GetFormattedSize(Size:Integer; SizeWidth:Byte;
UseSeparators:Boolean):String;
Var
  i, i3:Integer;
  Res:String;
Begin
  Str(Size:SizeWidth, Result);
  If Not(UseSeparators) Then Exit;
  If Length(Result)<=3 Then Exit;
  i3:=0;
  Res:=Result;
  Result:='';
  For i:=Length(Res) DownTo 1 Do
  Begin
    If i3=3 Then
    Begin
      Result:=ConstSizeSeparator+Result;
      i3:=0;
    End;
    Result:=Res[i]+Result;
    Inc(i3);
  End;
End;

Function GetFormattedDateTime(DateTime:TDateTime):String;
Begin
  Result:=DateTimeToStr(DateTime);
End;
```

```
Function GetFormattedAttr(Attr:Integer):String;
Begin
  Result:='--';
  If ((Attr And faReadOnly)>0) Then Result[1]:=ConstReadOnly;
  If ((Attr And faArchive)>0) Then Result[2]:=ConstArchive;
  If ((Attr And faHidden)>0) Then Result[3]:=ConstHidden;
  If ((Attr And faSysFile)>0) Then Result[4]:=ConstSystem;
End;

Procedure GetFormattedItem(Item:TFileRecord; Var
FormattedItem:TFileFormattedRecord);
Begin
  FormattedItem.Name:=GetFormattedName(Item.Name);
  FormattedItem.Ext:=GetFormattedExt(Item.Ext);
  If IsDirectory(Item) Then
  Begin
    FormattedItem.Size:=ConstDirectory;
  End
  Else Begin
    FormattedItem.Size:=GetFormattedSize(Item.Size, 1, True);
  End;
  FormattedItem.DateTime:=GetFormattedDateTime(Item.DateTime);
  FormattedItem.Attr:=GetFormattedAttr(Item.Attr);
End;

Procedure GetItemByFileName(FileName:String; Var Item:TFileRecord);
Var
  F:File;
Begin
  FillChar(Item, SizeOf(Item), 0);
  If FileExists(FileName) Then
  Begin
    Item.Name:=ExtractFileName(FileName);
    Item.Ext:=ExtractFileExt(FileName);
    If Item.Ext<>'' Then
    Begin
      Delete(Item.Name, Length(Item.Name)-Length(Item.Ext)+1, Length(Item.Ext));
      Delete(Item.Ext, 1, 1);
    End;
    Item.DateTime:=FileDateToDateTime(FileAge(FileName));
    Item.Attr:=FileGetAttr(FileName);

   {$ I-I-}
    AssignFile(F, FileName);
    Reset(F, 1);
    If IOResult=0 Then
      Item.Size:=FileSize(F)
    Else
      Item.Size:=0;
    Close(F);
   {$I+}
    Exit;
  End;

  Item.Name:=ExtractFileName(FileName);
  Delete(Item.Name, 1, 1);
  Delete(Item.Name, Length(Item.Name), 1);
  If DirectoryExists(ExtractFilePath(FileName)+Item.Name) Then
  Begin
    Item.Ext:='';
    Item.Size:=0;
    Item.DateTime:=0;
    Item.Attr:=faDirectory;
  End
  Else Begin
    Item.Name:='';
  End;
End;
```

```
Function GetItemImageIndex(Item:TFileRecord):Integer;
Var
  i:Integer;
Begin
  If Item.Name=ConstDirUp Then
  Begin
    Result:=10;
    Exit;
  End;
  If IsDirectory(Item) Then
  Begin
    Result:=11;
    Exit;
  End;
  Item.Ext:=UpperCase(Item.Ext);
  For i:=0 To MaxFileTypes-1 Do
  Begin
    If Item.Ext=Images[i, 0] Then
    Begin
      Result:=StrToInt(Images[i, 1]);
      Exit;
    End;
  End;
  Result:=19;
End;

Function GetCompactSize(Size:Int64):String;
Begin
  If Size<=ConstBytesLimit Then
  Begin
    Result:=GetFormattedSize(Size,1, True)+' '+ConstBytes;
    Exit;
  End;
  If Size<=ConstKBytesLimit Then
  Begin
    Result:=GetFormattedSize((Size Div 1024), 1, True)+' '+ConstKBytes;
    Exit;
  End;
  Result:=GetFormattedSize((Size Div (1024*1024)), 1, True)+' '+ConstMBytes;
End;

Function GetFullName(Item:TFileRecord):String;
Begin
  If Item.Ext='' Then
    Result:=Item.Name
  Else
   Result:=Item.Name+'.'+Item.Ext;
End;

end.
```

Файл FilesEx.pas - Обробка помилок

```
unit FilesEx;

interface
Uses
  SysUtils, FileCtrl, ShellApi, Windows;

Const
  F_ER_SUCCESS=0;
  F_ER_HIMSELF=-1;
  F_ER_EXISTS =-2;
  F_ER_NOT_EXISTS=-3;
  F_ER_DIREXISTS=-4;
  F_ER_NOTCOPY=-128;
  F_ER_ERROR=-255;

  ConstCopyToDir:String='';

Procedure GetDiskSize(CurrentDrive:Char; Var TotalBytes, TotalFree:Int64);
Procedure GetRealDiskSize(Drive:Char; Var TotalBytes, TotalFree:Double);

Function ExecuteOneFile(WorkDir, FileName, Params:String):Integer;
Function GetExecuteError(ErrorCode:Integer):String;

Function CopyOneFile(FromFile, ToFile:String; PrevCheck:Boolean):Integer;
Function GetFileError(ErrorCode:Integer):String;

Function RenameOneFile(OldName, NewName:String):Integer;
Function DeleteOneFile(FileName:String):Integer;
Function DeleteOneDir(FileName:String):Integer;

Function CreateOneFolder(FolderName:String):Integer;

implementation


function GetDiskFreeSpaceEx(lpDirectoryName: PAnsiChar;
              var lpFreeBytesAvailableToCaller : Integer;
              var lpTotalNumberOfBytes: Integer;
              var lpTotalNumberOfFreeBytes: Integer) : boolean;
              stdcall;
              external 'kernel32'
              name 'GetDiskFreeSpaceEx';

Procedure GetDiskSize(CurrentDrive:Char; Var TotalBytes, TotalFree:Int64);
Var
  Drive:Byte;
Begin
  CurrentDrive:=UpCase(CurrentDrive);
  Drive:=Ord(CurrentDrive)-64;
  TotalBytes:=DiskSize(Drive);
  TotalFree:=DiskFree(Drive);
End;

Procedure GetRealDiskSize(Drive:Char; Var TotalBytes, TotalFree:Double);
Var
  AvailToCall : integer;
  TheSize : integer;
  FreeAvail : integer;
  TheDrive:String;
Begin
  TheDrive:=Drive+':\'+#0;

  GetDiskFreeSpaceEx(@TheDrive[1], AvailToCall, TheSize, FreeAvail);
{$IFOPT Q+}
{$DEFINE TURNOVERFLOWON}
```

```
 {$ Q-Q-}
{$ENDIF}
  If TheSize >= 0 then
    TotalBytes := TheSize
  Else
    if TheSize = -1 then
    begin
      TotalBytes := $7FFFFFFF;
      TotalBytes := TotalBytes * 2;
      TotalBytes := TotalBytes + 1;
    end
    else begin
      TotalBytes := $7FFFFFFF;
      TotalBytes := TotalBytes + abs($7FFFFFFF - TheSize);
    end;

  If AvailToCall >= 0 then
    TotalFree := AvailToCall
  else
    if AvailToCall = -1 then
    begin
      TotalFree := $7FFFFFFF;
      TotalFree := TotalFree * 2;
      TotalFree := TotalFree + 1;
    end
    else begin
      TotalFree := $7FFFFFFF;
      TotalFree := TotalFree + abs($7FFFFFFF - AvailToCall);
    end;
End;

Function ExecuteOneFile(WorkDir, FileName, Params:String):Integer;
Begin
  FileName:=FileName+#0;
  WorkDir:=WorkDir+#0;
  Params:=Params+#0;

  Result:=ShellExecute(0, 'open', @FileName[1], @Params[1], @WorkDir[1],
SW_SHOWNORMAL);
End;

Function GetExecuteError(ErrorCode:Integer):String;
Begin
  Result:='';
  If ErrorCode>32 Then Exit;
  Case ErrorCode Of
    0                       : Result:='Системі не вистачає пам'яті або ресурсів';
    ERROR_FILE_NOT_FOUND    : Result:='Файл не знайдений';
    ERROR_PATH_NOT_FOUND    : Result:='Шлях не знайдений';
    ERROR_BAD_FORMAT        : Result:='Помилка у форматі файлу';
    SE_ERR_ACCESSDENIED     : Result:='Доступ до файлу закритий';
    SE_ERR_ASSOCINCOMPLETE: Result:='Файлова асоціація невірна';
    SE_ERR_DLLNOTFOUND      : Result:='Динамічна бібліотека не знайдена';
    SE_ERR_NOASSOC          : Result:='Відсутній додаток, пов'язаний з даним типом
файлу';
    SE_ERR_OOM              : Result:='Недостатньо пам'яті для завершення
операції';
    SE_ERR_SHARE            : Result:='Помилка спільного доступу';
  Else
    Result:='Помилка при запуску програми';
  End;
End;

Function CopyOneFile(FromFile, ToFile:String; PrevCheck:Boolean):Integer;
Begin
  If PrevCheck Then
  Begin
    If FromFile=ToFile Then
    Begin
```

```
      Result:=F_ER_HIMSELF;
      Exit;
    End;
    If FileExists(ToFile) Then
    Begin
      Result:=F_ER_EXISTS;
      Exit;
    End;


  End;

  FromFile:=FromFile+#0;
  ToFile:=ToFile+#0;

  If Not(CopyFile(@FromFile[1], @ToFile[1], False)) Then
    Result:=F_ER_NOTCOPY
  Else
    Result:=F_ER_SUCCESS;
End;

Function GetFileError(ErrorCode:Integer):String;
Begin
  Result:='';
  Case ErrorCode Of
    F_ER_SUCCESS: Result:='';
    F_ER_HIMSELF: Result:='Не можна копіювати файл у себе';
    F_ER_EXISTS : Result:='Такий файл уже існує';
    F_ER_DIREXISTS : Result:='Така папка вже існує';
    F_ER_NOT_EXISTS : Result:='Такий файл відсутній';
    F_ER_NOTCOPY: Result:='Помилка при копіюванні';
    F_ER_ERROR: Result:='Помилка при роботі з файлом';
  End;
End;

Function RenameOneFile(OldName, NewName:String):Integer;
Begin
  If (FileExists(NewName) Or DirectoryExists(NewName)) Then
  Begin
    Result:=F_ER_EXISTS;
    Exit;
  End;
  If MoveFile(PChar(OldName+#0), PChar(NewName+#0)) Then
    Result:=F_ER_SUCCESS
  Else
    Result:=F_ER_NOTCOPY;
End;

Function DeleteOneFile(FileName:String):Integer;
Begin
  If Not(FileExists(FileName)) Then
  Begin
    Result:=F_ER_NOT_EXISTS;
    Exit;
  End;
{$ I-I-}
  FileSetAttr(FileName, faArchive);
  IOResult;
{$I+}
  If SysUtils.DeleteFile(FileName) Then
    Result:=F_ER_SUCCESS
  Else
    Result:=F_ER_ERROR;
End;

Function DeleteOneDir(FileName:String):Integer;
Begin
  If Not(DirectoryExists(FileName)) Then
  Begin
```

```
    Result:=F_ER_NOT_EXISTS;
    Exit;
  End;
  If RemoveDir(FileName) Then
    Result:=F_ER_SUCCESS
  Else
    Result:=F_ER_ERROR;
End;

Function CreateOneFolder(FolderName:String):Integer;
Begin
  If DirectoryExists(FolderName) Then
  Begin
    Result:=F_ER_DIREXISTS;
    Exit;
  End;
  If CreateDir(FolderName) Then
    Result:=F_ER_SUCCESS
  Else
    Result:=F_ER_ERROR;
End;

end.
```

Файл about.pas - Довідка

```
unit about;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, jpeg, ExtCtrls;

type
  TFmAbout = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Image1: TImage;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FmAbout: TFmAbout;

implementation

{$R *.dfm}

procedure TFmAbout.FormCreate(Sender: TObject);
begin
Memo1.Clear;
Memo1.Lines.Add('БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА');
Memo1.Lines.Add('');
Memo1.Lines.Add('на  тему:');
Memo1.Lines.Add('');
Memo1.Lines.Add(' Програмне забезпечення ієрархічних систем зберігання даних з
використанням технології Tiered Storage');
Memo1.Lines.Add('');
Memo1.Lines.Add('');
Memo1.Lines.Add('Керівник: Усік П.С');
Memo1.Lines.Add('');
Memo1.Lines.Add('Розробив: студент Казаченко Владислав Ігорович ');
Memo1.Lines.Add('                    гр. КІ-21-3СК');
Memo1.Lines.Add('');
Memo1.Lines.Add('м. Кропивницький 2024');
Memo1.Lines.Add('');
end;

procedure TFmAbout.Button1Click(Sender: TObject);
begin
FmAbout.Close;
end;
end.
```