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
“____” _____ 2023 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи кібербезпеки універсальної системи архівування файлів для забезпечення цілісності”

Виконав здобувач вищої освіти
IV курсу, групи КБ-19
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Величко Є.Д.
«____» _____ 2023 р.

Керівник проекту
доктор технічних наук, професор
_____ Смірнов О.А.
«____» _____ 2023 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 *125 "*Кібербезпека*"*
Освітньо-професійна (освітньо-наукова) програма _*"*Кібербезпека*"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2023 року

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

Величку Євгенію Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки універсальної системи архівування файлів для забезпечення цілісності*

2. Керівник роботи *Смірнов Олексій Анатолійович, докт. техн. наук, професор*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 12-02 від 5.01.2023 року

3. Строк подання студентом роботи до захисту *23.05.2023 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи кібербезпеки універсальної системи архівування файлів для забезпечення цілісності*

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 кібербезпеки в промислову експлуатацію.
6. Висновки

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

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

7. Дата видачі завдання « 17 » січня 2023 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	10.03.2023 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2023 р.	
3.	Розробка моделі компонента	20.03.2023 р.	
4.	Розробка структур даних	25.03.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2023 р.	
6.	Програмування алгоритмів	10.04.2023 р.	
7.	Оформлення ПЗ	17.04.2023 р.	
8.	Попередній захист роботи	23.05.2023 р.	

Дата видачі завдання
« 17 » січня 2023 р.

Підпис керівника
_____ Смірнов О.А.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2023 р.

Підпис здобувача
_____ Величко Є.Д.
(прізвище та ініціали)

АНОТАЦІЯ

Величко Є.Д. Програмне забезпечення системи кібербезпеки універсальної системи архівування файлів для забезпечення цілісності. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи кібербезпеки універсальної системи архівування файлів для забезпечення цілісності.

Метою розробки є програмне забезпечення системи кібербезпеки універсальної системи архівування файлів для забезпечення цілісності.

Результат роботи – програмна реалізація системи кібербезпеки універсальної системи архівування файлів для забезпечення цілісності.

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 архітектури IBM PC з ОС Windows 10/11.

Програму розроблено в середовищі RAD Studio Delphi 10.4.

Ключові слова: кібербезпека, архівування файлів, цілісність

ABSTRACT

Velychko E.D. Universal file archiving system cybersecurity software for integrity. 125 Cyber security. Central Ukrainian National Technical University. Kropyvnytskyi. 2023.

In this graduation thesis for the first (bachelor) level of higher education, software is developed, which is intended for the cyber security system of the universal file archiving system to ensure integrity.

The goal of the development is the cybersecurity system software of a universal file archiving system to ensure integrity.

The result of the work is the software implementation of the cyber security system of the universal file archiving system to ensure integrity.

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 PCs of IBM PC architecture with Windows 10/11 OS.

The program was developed in the RAD Studio Delphi 10.4 environment.

Keywords: cyber security, file archiving, integrity

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

7z	–	формат заархівованого файлу
BCJ	–	попередня обробка файлів, що виконуються
Blu-ray	–	носій інформації
CD	–	носій інформації
DVD	–	носій інформації
LZH	–	формат заархівованого файлу
LZMA	–	алгоритм архівування
LZP	–	знаходження повторів в тексті
Rar	–	формат заархівованого файлу
REP	–	знаходження повторів
zip	–	формат заархівованого файлу

ВСТУП

Актуальність теми. На стиск даних багато користувачів не обертають уваги, але він буквально оточує нас: будь-який установочний пакет з новою програмою й багато форматів файлів, такі як JPEG для фотографій або різні формати аудіо й відео, використовують стиск (без втрат або із втратами), щоб заощадити місце на сховище або час передачі, а також і гроші. Користувачі звичайно вдаються до допомоги стиску файлів, коли вони зіштовхуються з деякими обмеженнями під час роботи з ними. Робота з одним файлом-архівом замість безлічі, а також високий рівень стиску часто бувають дуже актуальні. Втім, не менш важливий й час, що йде на стиск файлів і архівацію.

Програми архівування дозволяють, крім процедур архівації, виконувати дії пов'язані з кібербезпекою: перевіряти на наявність вірусів, перевіряти цілісність й т.д.

Незалежно від додаткових функцій, одним з головних параметрів є ефективність. Найкраще, якщо архіватор забезпечує дуже високий ступінь стиску, причому для максимально різноманітних типів файлів, та й швидкий час стиску теж немаловажний. Багато утиліт здатні працювати, принаймні, з популярним форматом zip, а деякі також підтримують архіви rar або LZH. Інші архіватори часто обіцяють більшу ефективність і більшу гнучкість.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи кібербезпеки універсальної системи архівування файлів для забезпечення цілісності.

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

– Огляд існуючих систем універсальної системи архівування файлів для забезпечення цілісності.

– Дослідження системи кібербезпеки універсальної системи архівування файлів для забезпечення цілісності.

– Програмна реалізація системи кібербезпеки універсальної системи архівування файлів для забезпечення цілісності.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі універсальної системи архівування файлів для забезпечення цілісності.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки універсальної системи архівування файлів для забезпечення цілісності, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Система призначена для архівування файлів з забезпеченням цілісності та кібербезпеки. Безпека архівних файлів дозволяє контролювати доступ до даних у архівних файлах. Наприклад, ви можете використовувати функцію захисту архівних файлів, щоб запобігти будь-якому доступу до даних у певній таблиці чи стовпці для більшості користувачів, одночасно надаючи доступ членам вибраних ролей до тих самих даних.

Кожен захищений архівний файл пов’язаний із визначенням доступу до файлу (FAD), яке є визначенням безпеки, яке містить список таблиць і стовпців, для яких визначено права доступу, і для кожної зазначеної ролі надає або забороняє права доступу до архівованих даних.

Для встановлення безпеки архівних файлів потрібен ACD (за замовчуванням або створений вами для цієї мети), який використовується як основа для ролей у FAD. Крім того, ви повинні використовувати програму конфігурації, щоб увімкнути захист архівних файлів.

Домен контролю доступу – це визначення безпеки, яке служить основою для всіх рівнів безпеки. Кожен каталог, для якого ініціалізовано Security, містить ACD із назвою (за замовчуванням), який неможливо видалити. Залежно від потреб вашого закладу ви можете створити додаткові ACD або використовувати лише ACD (за замовчуванням). Кожен ACD містить список ролей. Кожна роль представляє логічне групування облікових записів користувачів і груп у вашій мережі. Як правило, ви можете призначити імена ролей, щоб передати можливості облікових записів, представлених роллю. Прикладами імен ролей можуть бути «ГІСТЬ», «ЗВИЧАЙНИЙ» і «СУПЕР». Облікові записи користувачів і груп залежно від обставин зіставляються з однією або кількома ролями.

Специфікації ролі в (за замовчуванням) ACD посилаються на функціональну безпеку, якщо ввімкнено для каталогу Optim. Крім того,

специфікації ролей у (за замовчуванням) або іншому ACD посилаються на списки керування доступом (використовуються для захисту об’єктів) і визначення доступу до файлів (використовуються для захисту архівних файлів), а також призначені привілеї для доступу до об’єкта чи архівного файлу.

Список контролю доступу – це об’єкт, який служить основою для захисту об’єктів. Параметри ACL керують здатністю ролі виконувати дії (такі як читання, оновлення чи видалення) як з об’єктом, так і з ACL для об’єкта. Кожен ACD, File Access Definition і захищений об’єкт мають унікальний ACL.

Визначення доступу до файлу є основою безпеки архівних файлів. Усі архівні файли, створені за допомогою запиту архіву, який посилається на FAD, захищені FAD. ACD (за замовчуванням) вибірково надає та забороняє привілеї функціональної безпеки для ролей, щоб забезпечити відповідний доступ до інтерфейсу та функцій. Наприклад, учасник ролі, який, як очікується, виконує запит на архівування в Інтернеті, повинен мати привілей Викликати редактор запитів на архів, тоді як учасник ролі, який повинен створювати захищений запит на архівування, повинен мати права Створити запит на архів і Зв’язати запит на архів .

Захищені об’єкти (включно з ACD та визначеннями доступу до файлів) мають ACL, який надає та забороняє дозволи на читання, оновлення, видалення та виконання для підмножини ролей, визначених у ACD (за замовчуванням). (На ваш вибір ці ролі можна визначити в спеціалізованому ACD, а не в ACD (за замовчуванням).) На ілюстрації списки ACL для запитів на архівування та відновлення мають надавати дозвіл на запуск (виконання) ролям, які, як очікується, виконуватимуть ці запити.

Визначення доступу до файлу (FAD) визначає правила безпеки для даних в одному або кількох архівних файлах, створених за допомогою архівного запиту, який посилається на FAD. FAD на діаграмі може надавати доступ до архівних даних у вибраних таблицях і стовпцях і забороняти доступ до даних в інших. Обліковий запис для входу, який використовується для запуску запиту на відновлення, має бути представлений роллю в FAD, якій надано необхідний доступ до архівованих даних.

Щоб використовувати Security, ви повинні ініціалізувати безпеку для каталогу, призначити адміністратора безпеки та ввімкнути функції безпеки, які використовуватиме ваш сайт.

1.2 Область застосування

Областю застосування є архівування даних. Розглянемо її більш детально. Для групи невеликих файлів, які необхідно відправити по електронній пошті, високий ступінь стиску не так і важливий, але має сенс об'єднати всі ці файли в один архів, оскільки працювати з ним буде зручніше. Тим більше що багато комерційних додатків забезпечують високий рівень інтеграції в операційну систему, що дозволяє вибирати всі необхідні функції простим клацанням правої клавіші миші по файлах або папкам в Windows Explorer. Найкраще вибирати популярні контейнери, такі як zip або rar, якщо необхідно відправити дані людям, яких не дуже добре знаєте. Швидше за все, багато користувачів зможуть відкрити й розпакувати файли zip або rar, чого не можна сказати про інші менш популярні формати.

Ситуація інша, якщо хочете зменшити розмір файлу. У цьому випадку можна спробувати знайти не таку популярну, але потужнішу утиліту. Втім, при цьому має сенс ознайомитися й з історією утиліти, наскільки регулярно вона обновлялася й чи будуть доступні відновлення, коли перейдете на нову операційну систему. Крім того, не перешкодить вибирати таку утиліту, чий контейнер можуть зчитувати й розпаковувати інші програми.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки універсальної системи архівування файлів для забезпечення цілісності, 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

Проведемо дослідження сучасних архіваторів. Мною розглянуті архіватори 7zip, FreeArc, WinRAR і WinZip, а також проаналізувано рівень і час стиску. Звичайно, у нас немає можливості протестувати всі доступні архіватори й утиліти, тому сконцентрувався на самих популярних (за інформацією download.com). Лідерами сьогодні є WinRAR і WinZip. Але також взяв 7zip і FreeArc.

Особливості архівування

Є деякі фактори, які варто брати до уваги при виборі підходящого архіватора.

Рівень стиску, швидкість і типи даних

Кращий рівень стиску дозволяє знизити розмір файлу, але час архівування може значно збільшуватися. Сучасні архіватори, як правило, оптимізовані під багатопоточність, тобто вони можуть вигравати від дво- або чотирьохядерних процесорів. Втім, як і раніше існують деякі програми, які навантажують тільки одне обчислювальне ядро. Важливо також розуміти, що деякі типи даних, такі як документи або файли з великим числом повторюваних ділянок, можна стиснути досить сильно, хоча інші дані стискати взагалі не має змісту, щоб не витрачати зайвий час (їх варто просто копіювати в архів). Фотографії JPEG, установочні файли програм і інші подібні дані вже стиснуті, тому не зможете повторно серйозно знизити розмір файлу. Як правило, виходить незначно зменшення розміру файлу при архівування фотографій або настановних пакетів.

Швидкість обробки є вторинною, якщо вам просто потрібно запакувати файли для відправлення по електронній пошті, але вона стає вже більш важливою, якщо вам потрібно архівувати великий обсяг даних. Наприклад, архівація 50 Гбайт різних даних може виконуватися так само швидко, як просте копіювання файлів, якщо обмежити себе мінімальним рівнем стиску, але вона може зайняти кілька годин на звичайному комп'ютері, якщо вибирати максимальний режим стиску.

Багатотомні архіви

Через обмеження накопичувачів або поштових серверів, часто потрібно обмежити розмір файлів або резервних контейнерів. Як правило, облікові записи електронної пошти дотепер обмежений 10 або 20 Мбайт на лист, та й такі носії, як CD, DVD або навіть диски Blu-ray, теж далеко не нескінченні. У подібних випадках має сенс створювати архів з декількох частин (томів), у якому можна задавати розмір кожної частини. Так звані багатотомні архіви підтримуються багатьма сучасними утилітами архівування, хоча й не всіма.

Підтримка командного рядка

Якщо необхідно автоматизувати архівацію або інтегрувати утиліти стиску в існуючі скрипти, тоді приємна новина: майже всі рішення, відомі нам, підтримують роботу через командний рядок. Всі чотири програми, розглянуті в цьому розділі, надають повноцінну підтримку командного рядка.

Паролі й шифрування

Щоб до ваших даних не добрався сторонній, найкраще захищати вміст архіву паролем і шифруванням. Звичайно, при цьому на стиск і шифрування піде більше часу, але зате така комбінація дає досить надійний рівень захисту. Більшість утиліт архівування підтримують шифрування AES, а деякі (7zip, WinZip) уже можуть виграти від додаткових процесорних інструкцій прискорення шифрування, які Intel початку впроваджувати разом із дводерними 32-розрядними процесорами Core i5. У підсумку, шифрування не буде робити помітного впливу на час обробки в майбутньому, якщо утиліти стиску

підтримують інструкції AES-NI. Що стосується AMD, то напевно лише справа часу, коли подібні інструкції з'являться в процесорах цієї компанії.

Додаткові обмеження

Втім, є ще деякі обмеження, зв'язані, як правило, з великими розмірами файлів і старих контейнерів. Zip у специфікації 2.0 підтримує розмір архіву тільки до 2 Гбайт, та й один окремий файл може бути не більше 2 Гбайт. З розділами вашого жорсткого диска можуть спостерігатися схожі проблеми: FAT32 на старих системах Windows не розуміє файли більше 4 Гбайт. Якщо буде потрібно створити більший архів, то прийдеться переходити на NTFS. Нарешті, сучасні утиліти архівування можуть споживати велику кількість пам'яті, тому що великі словники (де втримуються повторювані фрагменти) звичайно перебувають в оперативній пам'яті. Тому кращу продуктивність одержемо тільки тоді, коли утиліта архівування не буде впиратися в обсяг доступної пам'яті.

7zip

7zip – утиліта архівування з відкритим вихідним кодом, скачати яку можна на сайті 7zip.org. Вона підтримує розпакування різних форматів, а також кілька форматів для стиску. На web-сайті заявлено, що 7zip стискає у формат zip на 2-10% краще, ніж інші програми. Мається на увазі рівень стиску в порівнянні з рівнями стиску в опублікованому порівнянні шести різних утиліт архівування, що використовують формат zip.

Втім, реальною перевагою 7zip можна все-таки назвати власний формат за назвою 7z. Архітектура формату відкрита, вона дозволяє додавати 256-бітне шифрування файлів AES. Шифрування AES підтримується форматами 7z і zip. У випадку 7z можна не виявити більших переваг по продуктивності, незважаючи на наявність нових інструкцій: справа в тому, що алгоритм 7z уже досить "важкий", він уже значно сповільнює процес архівування, так що прискорення AES позначається слабко.

Формат 7zip не дуже сильно розповсюджений, але однаково він підтримується іншими утилітами архівування, такими як IZArc, PowerArchiver,

TUGZip і WinRAR. Як правило, 7zip використовує стиск LZMA (LZ77), але є можливість застосування інших алгоритмів. LZMA може працювати зі словниками різного розміру аж до 4 Гбайт (вимагає 64-бітну ОС). 7zip 9.1 також підтримує й більш просунутий алгоритм LZMA2.7zip може працювати з командним рядком, або можна використовувати стандартний графічний інтерфейс Windows, у який вбудовується утиліта. Доступно цілих 74 локалізованих язикових версій утиліти, та й програма може працювати під різними версіями Windows від Windows 98 до Windows 10/11. На відміну від WinZip, що залишається однією із самих популярних утиліт стиску, 7zip повністю оптимізована під багатопоточність, тому в достатньому ступені виграє від декількох обчислювальних ядер.

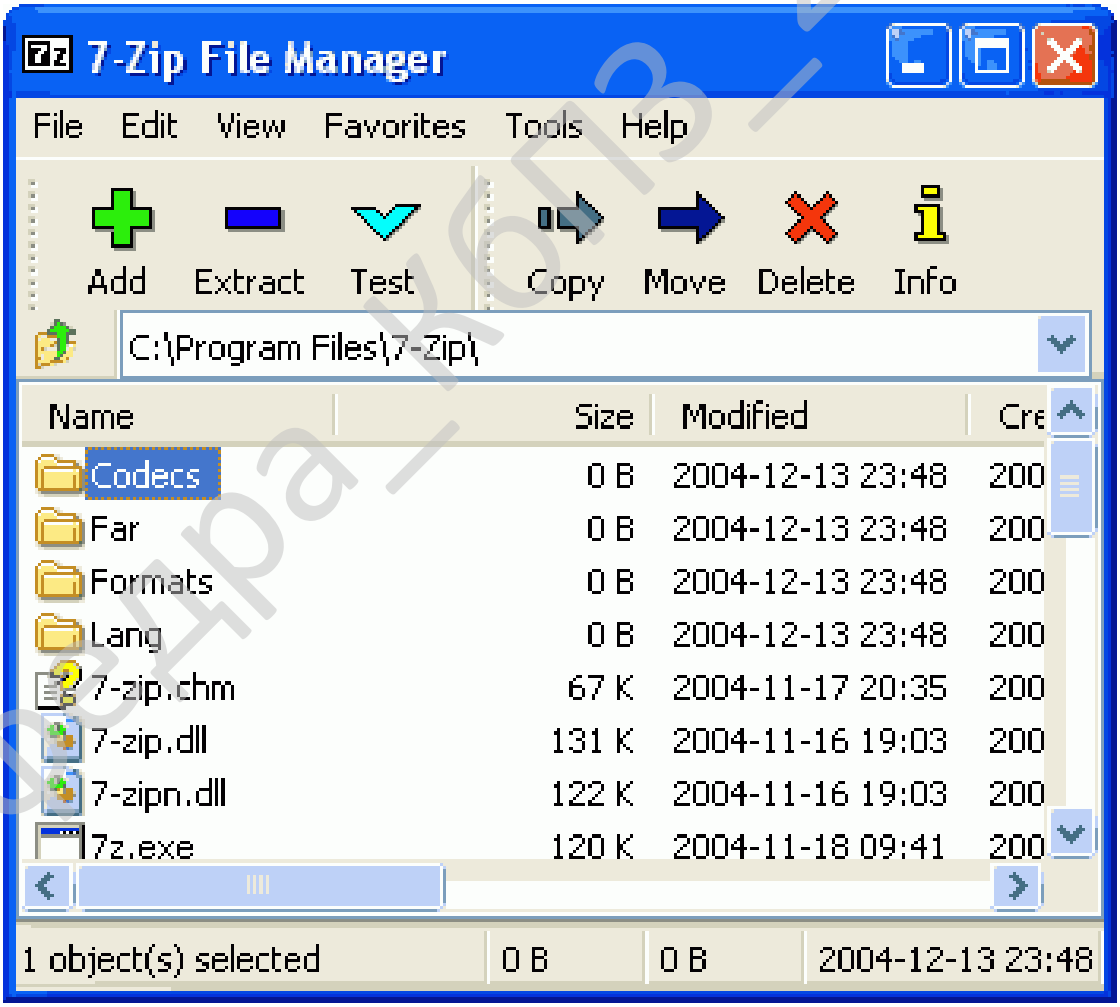

Рисунок 2.1 – Інтерфейс користувача архіватора 7zip

FreeArc

Утиліту архівування FreeArc можна скачати із сайту freearc.org, при цьому вона описується як сучасний архіватор, швидкий, але ефективний. На сайті говориться про те, що ця утиліта може працювати аж до 3х раз швидше, ніж кращі програми архівування – імовірно, автори мають на увазі WinZip і інші популярні продукти. Утиліта може працювати через командний рядок або через графічний інтерфейс, а список переваг на сайті вражає: програма навіть здатна перемикатися між різними алгоритмами стиску залежно від типу файлу. Звичайно, це допомагає максимізувати ступінь стиску, але також прив'язує вас до архіватору FreeArc, оскільки інші рішення навряд чи зможуть розпізнати або розпакувати ваші архіви. Якщо судити по web-сайту, то підтримуються 11 алгоритмів. Втім, на сайті не вказується, що всі ці алгоритми відносяться до стиску – вони можуть стосуватися читання або розпакування архівів. Можна також підключати зовнішні програми для обробки або фільтрації даних.

У межах обсягу до 1 Гбайт програма здатна знаходити повтори (REP), повтори в тексті (LZP), попередньо обробляти файли, що виконуються (BCJ), та й FreeArc може сортувати файли таким чином, щоб максимально використовувати перевагу від подібних функцій. Швидкий режим стиску автоматично приведе до використання GRZIP для текстів і Tornado для двійкових даних.

FreeArc також може додавати шифрування, щоб захищати ваші архіви. Підтримуються алгоритми AES, Blowfish, Twofish і Serpent. Також є підтримка архівів, що саморозпаковуються, об'єднання архівів, пережаття, блокування й додавання коментарів. Втім, зберігаються не всі атрибути файлів. Крім того, як і раніше не випущено 64-бітної версії FreeArc, та й підтримка багатотомних архівів відсутня. Дана утиліта добре підійде для тих, хто не проти витратити небагато часу й зусиль, щоб підбудувати архіватор під власне оточення. Розроблювачі планують пізніше додати повну підтримку форматів rar і 7z.

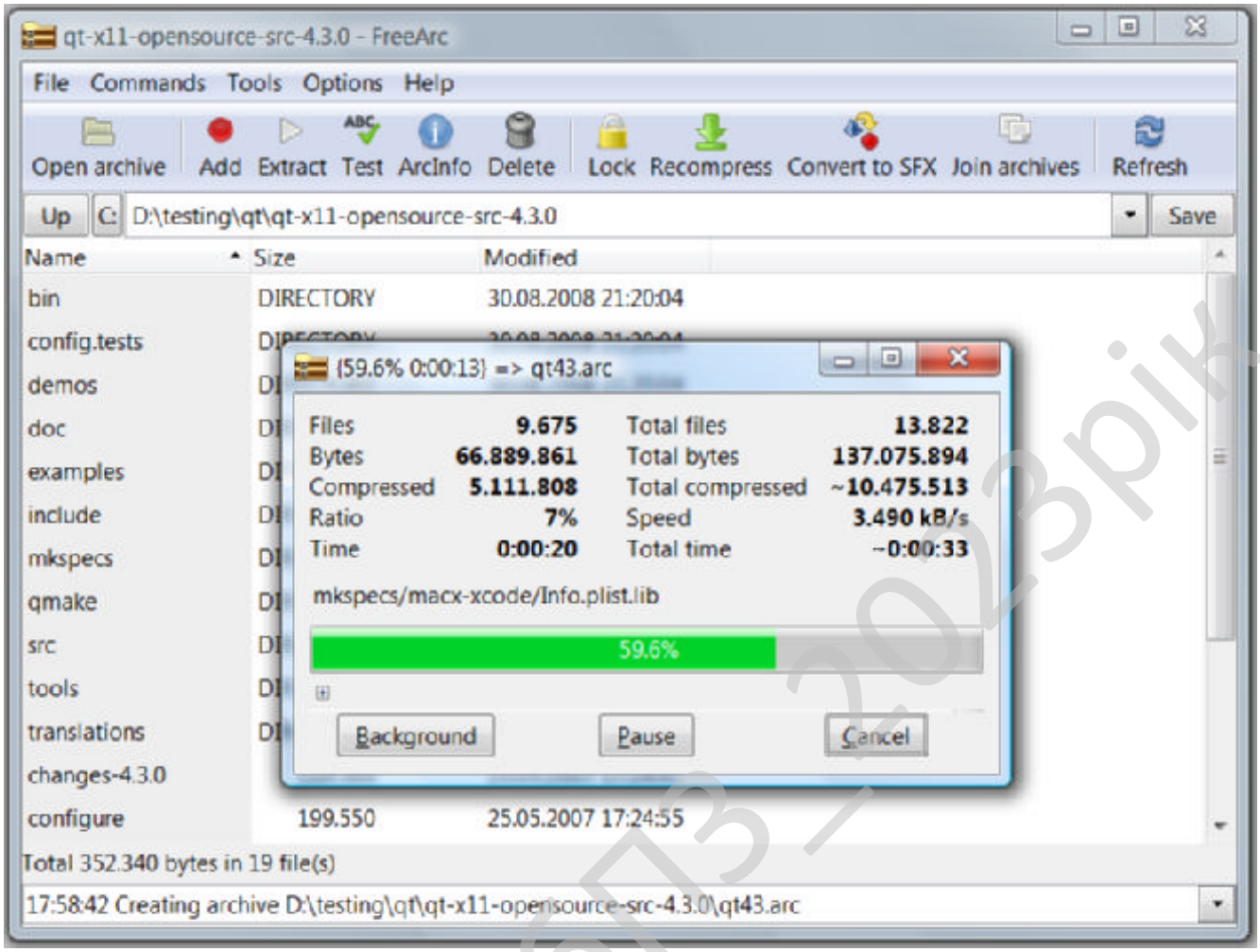

Рисунок 2.2 – Інтерфейс користувача архіватора FreeArc

WinRAR

Це одне із самих популярних рішень архівування з підтримкою багатопоточності. Утиліта WinRAR повністю підтримує архіви rar і zip, а також може розпаковувати архіви CAB, ARJ, LZH, TAR, GZ, ACE, UUE, BZ2, JAR, ISO, 7Z і Z. Подібно іншим утилітам, вона також підтримує командний рядок або працює через графічний інтерфейс, що підійде як ентузіастам, так і звичайним користувачам. WinRAR підтримує створення архівів, що саморозпаковуються, і шифрування архівів за допомогою алгоритму AES із ключем до 128 біт.

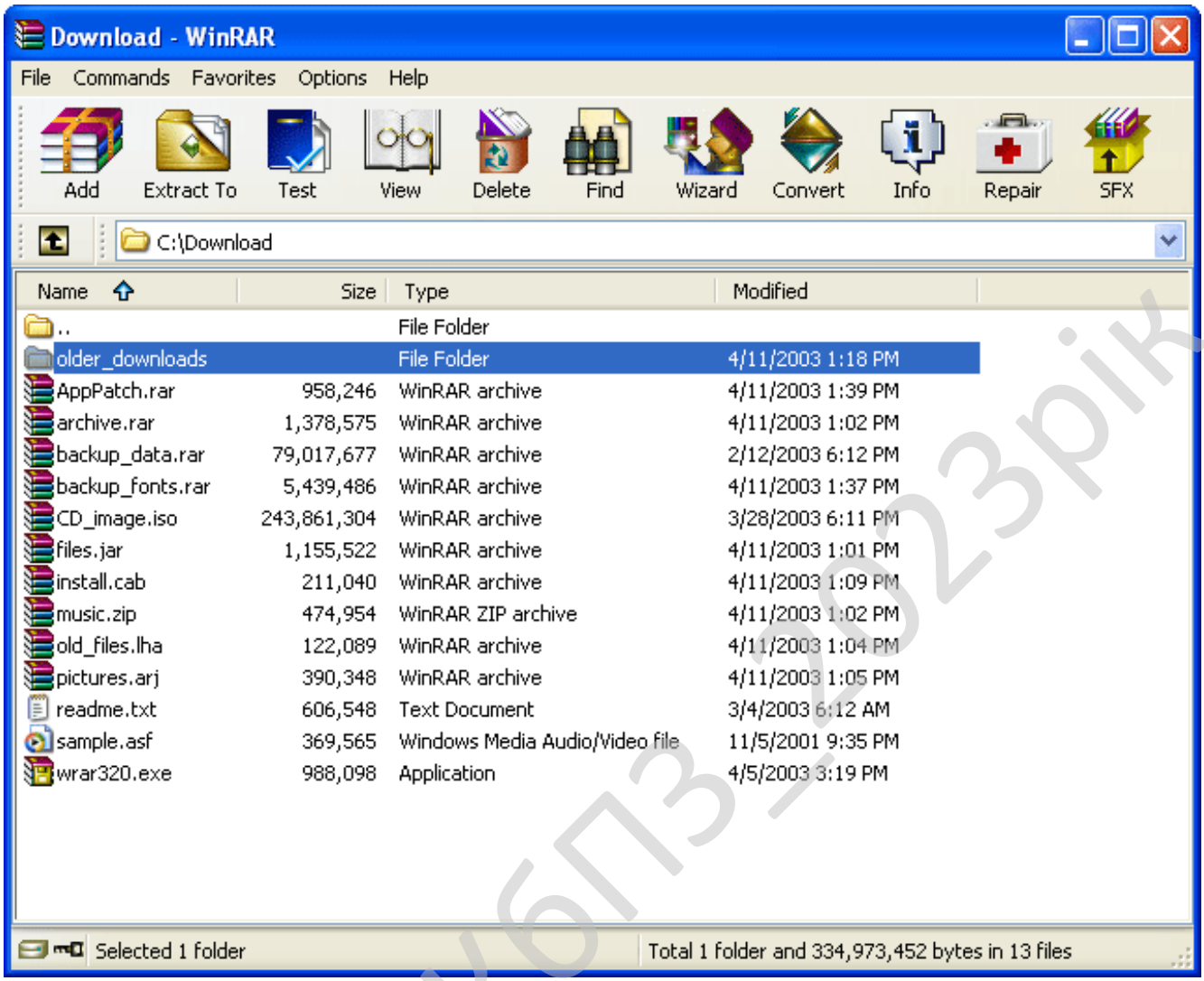

Рисунок 2.3 – Інтерфейс користувача архіватора WinRAR

WinRAR поставляється з Майстром архівування – приємне доповнення для новачків, оскільки він допоможе впоратися з усім крок за кроком. WinRAR також підтримує створення багатотомних архівів. Це може здатися не таким важливим, але ви по достоїнству оціните цю функцію, коли вам буде потрібно відіслати великий обсяг даних по електронній пошті, а поштова скринька адресата приймає листи строго обмеженого розміру. У такому випадку WinRAR можна використовувати, щоб розбити архів на кілька частин, які ви легко зможете відіслати окремо. Для WinRAR існує велика кількість аддонів, які

дозволяють поліпшити функціональність утиліти під різними операційними системами.

Після версії 3.91 WinRAR підтримує формат 7z з алгоритмом LZMA2 (розпакування), що може теж виявитися на руку, оскільки цей формат показує прекрасні результати стиску. Існують 32- і 64-бітні версії архіватора під різні операційні системи, але вони мало впливають на продуктивність. Поки що WinRAR доступний в 15 різних мовах. Нарешті, відзначимо, що можна міняти інтерфейс користувача утиліти за допомогою різних тим.

WinZip

Нарешті, у порівнянні бере участь утиліта WinZip, яку можна скачати на сайті winzip.com. Ця утиліта дуже популярна (особливо в англомовних країнах), але в наших тестах вона виявилася двоякою. З одного боку, вона добре інтегрується в операційну систему, користуватися утилітою дуже легко. З іншого боку, WinZip як і раніше не вміє розподіляти навантаження по декількох обчислювальних ядрах за допомогою потоків. Іншими словами: утиліта працює жахливо повільно при використанні LZMA або максимального рівня стиску, а зі звичайним рівнем стиску zip вона займає середню позицію. При виборі максимального рівня стиску утиліта вибирає оптимальний алгоритм залежно від типу файлу, якому потрібно заархівувати.

WinZip може створювати три різних типи архівів: zip, zipx і LHA. Але розпаковувати файли утиліта може з набагато більшого списку форматів: rar, 7z, BZ2, JAR, образи (img, iso) і файли cab. Якщо ви працюєте з об'ємними архівами в zip, то напевно зрадієте тому, що WinZip може створювати архіви zip більше 4 Гбайт. Програма також дозволяє користувачам додавати 128- або 256-бітне шифрування AES для забезпечення захисту даних. А функція автоматичного стирання гарантує, що тимчасово витягнуті файли із зашифрованих архівів будуть відразу ж віддалятися. Відзначимо й те, що WinZip може прискорюватися використанням нових інструкцій Intel AES на деяких процесорах Core i5 (всі настільні й більшість мобільних версій).

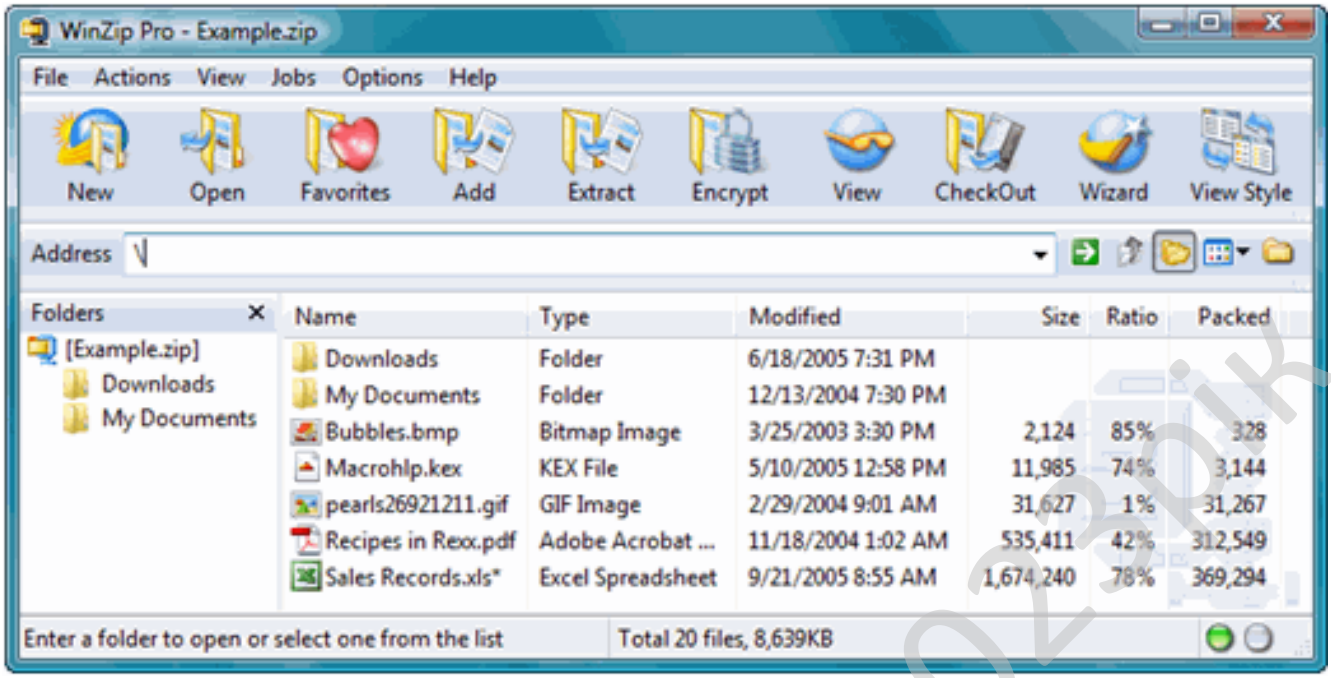

Рисунок 2.4 – Інтерфейс користувача архіватора WinZip

Не можливо перевірити всі функції WinZip, оскільки їхній набір великий. Підтримка Windows 10/11 сама по собі вже досить повна, оскільки утиліта підтримує жести на сенсорних дисплеях, попередній перегляд в Explorer, jump-списки для полегшення доступу до функцій zip і файлам. Остання версія також підтримує зміну розрішення фотографій через інтеграцію в Explorer перед тим, як стиснути їх в архів для відправлення по електронній пошті. Інші утиліти теж дають високий рівень інтеграції, але набір функцій не такий великий. WinZip може розбивати архіви zip на кілька томів, а убудований клієнт FTP може автоматично завантажувати архівні копії на зовнішній ftp-сервер (версії Backup і Pro). Є підтримка командного рядка, а також і створення архівів, що саморозпаковуються.

Результати проведених тестів

Власні формати: рівень стиску, розмір і час

Спочатку перевіримо власні формати, які підтримуються всіма чотирма архіваторами. Обсяг вихідних файлів становить 650 Мбайт. Вибиремо рівень стиску за замовчуванням, а також протестуємо найкращі настроювання стиску, які можливо виявити в графічному інтерфейсі утиліти.

Розмір архіву

Якщо подивитися на розмір архівів, то можна помітити досить вражаючу економію займаного простору. Цілком зрозуміло, що алгоритм стиску LZMA, а також і формат arc в FreeArc працюють найбільше оптимально, розмір архіву виходить від 350 до 356 Мбайт.

Рівень стиску

Рівень стиску показує, наскільки менше став архів, що вийшов. Архів став від 39,5% менше до 46,2%.

Час стиску

Час, що пішло на одержання архіву, теж немаловажний. Якщо WinZip треба було від трьох з половиною до чотирьох мінут, то в 7Zip з алгоритмом LZMA2 і рівнем стиску за замовчуванням пішло всього 45 секунд на виконання такої ж роботи. Цілком очевидно, що час стиску істотно розрізняється. У висновку помножу час стиску на розмір, файлу, що вийшов, це дасть загальний індекс продуктивності, що враховує обоє фактора.

Формат zip: рівень стиску, розмір і час

Тепер повторю той же самий тест, але всі утиліти будуть використовувати звичний формат zip. Це важливо, оскільки багато користувачів бажають продовжувати працювати з форматом zip із причин сумісності, та й будь-який користувач, навіть не дуже технічно грамотний, знаком з форматом zip. Я не зміг використовувати утиліту FreeArc, оскільки вона не підтримує стиск zip.

Розмір архіву

У цьому випадку розмір архіву набагато перевищує такий при використанні власних форматів стиску. Крім того, різниця між самим маленьким і більшим розміром архіву набагато менше. Утиліта 7zip показала себе дуже добре, оскільки ви не одержите істотної різниці при використанні стиску за замовчуванням або максимальним рівнем стиску. WinZip з максимальним рівнем стиску теж непогано показала себе.

Час стиску

Утиліта 7zip виконує стиск у режимі за замовчуванням у вісім разів швидше, ніж у максимальному режимі стиску. Результати WinZip і WinRAR перебувають посередині. Давайте подивимося на фінальні результати ефективності, які враховують рівень стиску й час.

Висновок за результатами

Розмір архіву, помножений на час архівування

Представимо підсумковий результат для власних форматів архіваторів. Мною помножений час архівування на розмір архіву, що вийшов. Чим менше обидві складові, тим нижче підсумковий результат. Цілком очевидно, що архіватор WinZip, що є найбільш популярною й дружньою до користувачів утилітою, виявився в цьому тесті в аутсайдерах. І вся справа в тому, що архіватор дотепер є однопотоковим додатком. Якби розроблювачі витратили достатній час, щоб оптимізувати свій продукт, то результат міг би бути набагато краще.

WinRAR дає прекрасні результати в обох режимах, але явним переможцем можна назвати утиліту 7zip у режимі стиску LZMA2 за замовчуванням. Якщо хочете сполучити швидку архівацію з високим рівнем стиску, то це саме такий випадок – утиліта найкраще показала себе з нашим тестовим набором на 650 Мбайт із різних файлів.

Розмір архіву, помножений на час архівування

Давайте глянемо на результати zip, які теж важливі для багатьох користувачів, у яких немає можливості вибору формату архівування. Знову ж,

утиліта 7zip дала високий рівень стиску при дуже маленькому часі виконання завдання. Утиліти WinRAR і WinZip явно відстають. Максимальний рівень стиску помітно збільшує час обробки, особливо в 7zip.

Висновок

Мною розглянуто чотири утиліти архівування, а також проведено тести стиску 650-мбайт набору різних файлів. У першій серії тестів кожний архіватор використовував власний формат стиску, а саме 7z з LZMA2 для 7zip, arc для FreeArc, rar для WinRAR і zip/best для WinZip. У другій серії тестів зрівняв результати стиску в популярний формат zip, що важливий для більшості користувачів.

Результати досить цікаві, оскільки спостерігаються серйозні розходження за часом обробки й не менш серйозні розходження по розміру архіву, що виходить. Утиліти arc і 7zip/LZMA2 забезпечили мінімальний розмір архіву нашого тестового набору при виборі максимального рівня стиску. На жаль, при цьому arc і LZMA віднімають досить істотний час. Якщо необхідно сполучити високий рівень стиску із прийнятним часом обробки, то рекомендую алгоритм LZMA2 утиліти 7zip зі штатними настроюваннями.

Якщо використовувати zip у різних утилітах, то підсумковий розмір архіву не буде істотно розрізнятися. Архіватори 7zip і WinZip дали найвищий ступінь стиску при виборі максимального режиму. Однак час обробки в обох випадках був занадто довгим.

Якщо можна вільно вибирати формат архіватора, і якщо необхідно одержати баланс між рівнем стиску й часом обробки, то 7zip з алгоритмом LZMA2 і WinRAR (обидва варіанти при штатних настроюваннях) зможуть дати найкращий результат у цілому. Якщо ж не можна піти з формату zip, то краще вибирати, знову ж, 7zip або WinRAR, і знову на штатних настроюваннях. Якщо виставляти максимальний стиск, то імовірно, не хвилює час обробки, тому щодо цього не буду давати яких-небудь рекомендацій.

Залишається відзначити, що утиліти WinRAR і WinZip стали переможцями по дружньому до користувача інтерфейсу. Набір функцій, особливо в WinZip, дуже багатий – від нього зможуть чимало виграти технічно менш підковані користувачі. WinZip забезпечує максимальний набір функцій, а WinRAR надає спеціального Майстра для покрокового виконання дій. Фахівців навряд чи збентежать утиліти 7zip і FreeArc.

Мною підібраний тестовий набір з різними типами файлів, але цілком можливо, що після тонкого настроювання кожного архіватора можна одержати кращі результати. Втім, з погляду продуктивності залишається тільки поскаржитися на відсутність в WinZip підтримки багатопоточності. Це єдина утиліта, що як і раніше опирається на одне обчислювальне ядро в часи, коли шестиядерні CPU уже будуть незабаром у продажі.

2.2 Обґрунтування вибору засобів для побудови системи кібербезпеки 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 системи кібербезпеки універсальної системи архівування файлів для забезпечення цілісності.

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 кібербезпеки 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 кібербезпеки 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

Проведемо опис найпопулярніших алгоритмів стиснення даних. Алгоритми стиснення даних можна визначити як процес зменшення розмірів файлів під час збереження тих самих або схожих певною мірою даних. Це робиться шляхом видалення непотрібних даних або повторного створення даних для більшої ефективності.

Під час стиснення у вас є можливість вибрати метод із втратами чи без втрат. Якщо ми говоримо про метод із втратами даних, він назавжди видаляє дані. З іншого боку, без втрат подбає про ваші вихідні дані. Вибраний тип залежить від того, якої якості ви хочете мати файли.
У цій статті ви знайдете суміш алгоритмів стиснення даних без втрат і алгоритмів стиснення зображень і відео на основі глибокого навчання.

Алгоритми стиснення даних без втрат зазвичай використовуються для виконання функції архівування або будь-яких інших високоякісних функцій. Ці алгоритми стиснення даних дозволяють зменшити розмір файлу. Це також гарантує повне відновлення файлів, якщо їх потрібно було відновити.

LZ77

LZ77 був оголошений у 1977 році та названий основою багатьох інших алгоритмів стиснення без втрат. Зазвичай він використовує метод « ковзного вікна ». У цій методології він піклується про словник, який використовує трійки для представлення:

– Зсув – це можна назвати фактичним початком фрази та початком файлу.

– Довжина – визначається як кількість символів, які допомагають вам у створенні фрази.

– Відхиляються символи – це маркетологи, які вказують нову фразу.

Він містить вказівку на те, що використана фраза повністю відповідає вихідній фразі, а також визначає, чи є інший символ.

Під час аналізу файлу словник динамічно оновлюється для відображення вмісту та розміру стиснутих даних.

LZR

LZR був розроблений і анонсований у 1981 році. Майкл Роде оголосив про нього, а пізніше він його змінив. Його можна використовувати як основну альтернативу для алгоритму стиснення даних LZ77, але ви також можете використовувати його для будь-якого зміщення у файлі. Якщо ви не використовуєте його лінійно, тоді йому потрібен значний обсяг пам’яті. Ця умова робить LZ77 кращим варіантом для використання. Цей алгоритм стиснення даних простий у застосуванні та має потенціал для дуже високої продуктивності, якщо його реалізувати на апаратному забезпеченні.

Це алгоритм, який широко використовується в утиліті компресії алгоритму стиснення даних Unix і використовується у форматі зображень GIF. Він став першим алгоритмом стиснення даних, який широко використовувався на комп'ютерах. Великий текстовий файл англійською мовою зазвичай можна стиснути з LZW приблизно до половини початкового розміру.

LZSS

LZSS розшифровується як Lempel Ziv Storer Szymanski, і він був розроблений і оголошений у 1982 році. Це алгоритм стиснення даних, який покращує LZ77. Цей процес стиснення виконується шляхом включення методу, який стежитиме за тим, чи заміна зменшує розмір файлу. Якщо воно не зменшиться, то введення буде залишено у вихідному вигляді. Він також не надає перевагу використанню символів, що відхиляються, і надає перевагу лише використанню пар довжини зсуву.

Цей алгоритм в основному використовується для архівування файлів у різні формати, такі як RAR або ZIP, або для стиснення мережевих даних.

Це можна назвати технікою словникового кодування. LZSS намагається замінити рядок символів посиланням на розташування словника того самого рядка. Ключова відмінність між LZ77 і LZSS полягає в тому, що в LZ77 посилання на словник може бути довшим за рядок, який воно замінює. У LZSS такі посилання пропускаються, якщо довжина менша за точку «беззбитковості».

Deflate

Deflate – це формат файлу алгоритму стиснення даних без втрат, у якому використовується комбінація кодування LZSS і Хаффмана. Він був розроблений Філом Катцом у 1993 році. Кодування Хаффмана також є алгоритмом, розробленим у 1952 році. Його можна визначити як алгоритм ентропійного кодування, який допомагає вам призначати свій код на основі частоти символу.

Кац також розробив оригінальний алгоритм, який використовується для побудови потоків Deflate. Як було зазначено в документі RFC, вважалося, що алгоритм створення файлів Deflate можна реалізувати таким чином, що не охоплюється патентами. Це призвело до його широкого використання на додаток до формату файлу ZIP, який був основною метою Katz для його розробки. Патент більше не доступний.

LZMA

LZMA розшифровується як ланцюговий алгоритм Лемпеля Зіва Маркова, і він був розроблений і випущений у 1998 році. Його також можна назвати модифікацією LZ77. Ця модифікація була зроблена для архіватора –Zip із форматом .7z. Використовується метод ланцюгового стиснення, який реалізує модифікований LZ77 на бітовому, а не на байтовому рівні. Вихідні дані будуть додатково оброблені за допомогою арифметичного кодування для більшого стиснення.

Продуктивність інших етапів стиснення залежить від точної реалізації. Цей алгоритм стиснення даних використовує схему стиснення словника, дещо дуже схожу на алгоритм LZ77, опублікований Абрахамом Лемпелем і Якобом Зівом у 1977 році. Він також має високий коефіцієнт стиснення та змінний розмір

словника стиснення. Хоча він все ще підтримує швидкість декомпресії, дуже подібну до інших широко використовуваних алгоритмів стиснення.

LZMA2

LZMA2 був розроблений і випущений в 2009 році. Його також можна визначити як модифікацію LZMA. Це покращує LZMA, покращуючи його продуктивність за допомогою більших можливостей багатопоточності. LZMA2 також надає вам покращену обробку нестисливих даних. Це простий контейнерний формат, який може включати як нестиснуті дані, так і дані LZMA, а також із кількома різними параметрами кодування LZMA.

LZMA2 підтримує довільне масштабоване багатопотокове стиснення та декомпресію, а також ефективне стиснення даних, які є частково нестиснутими. Однак він може мати певні проблеми з безпекою, бути небезпечним і менш ефективним, ніж LZMA. Простіше кажучи, це може бути корисним для вас, але так, це не настільки безпечно порівняно з LZMA.

Стиснення на основі багаторівневого перцептрона (MLP).

MLP можна визначити як технологію, яка використовує декілька шарів нейронів для введення, обробки та надання вихідних даних. Він може бути реалізований для зменшення розмірності завдань, а також для стиснення даних. Алгоритм на основі MLP був вперше розроблений у 1988 році та приєднався до вже існуючих процесів:

– Двійкове кодування – стандартне двосимвольне кодування.

– Квантування – проблеми введення з неперервної множини в дискретну множину.

– Трансформація просторової області – піксель за пікселем змінює дані.

Для визначення оптимального двійкового коду алгоритм MLP використовує вихідні дані вищевказаних процесів у нейронну мережу декомпозиції.

Крім того, цей алгоритм був модифікований за допомогою інтуїтивно зрозумілих методів, які дозволяли точну апроксимацію даних повністю на основі

сусідніх даних через зворотне поширення. Це може бути дуже корисним для стиснення даних зображень і відео.

Dee Coder – стиснення відео на основі глибокої нейронної мережі

Deep Coder визначається як структура на основі згорткової нейронної мережі (CNN). Це робить подання альтернативою тим методам стиснення відео, які ми використовуємо так довго. Для прогнозних і залишкових сигналів ця модель використовує різні згорткові нейронні мережі (CNN).

Він виконує кодування карт ознак у двійковий потік із використанням скалярного квантування та дуже старого традиційного алгоритму стиснення файлів під назвою кодування Хаффмана. Стверджується, що ця модель здатна забезпечити вищу продуктивність у порівнянні з відомим стандартом кодування відео H.264/AVC.

Згорточна нейронна мережа (CNN)

CNN визначаються як нейронні мережі різних рівнів. Це в основному використовується для розпізнавання зображень і виявлення ознак. Коли ви застосовуєте його до стиснення, ці мережі використовують згортку для обчислення зв’язку між сусідніми пікселями. Якщо порівнювати його з алгоритмами на основі MLP, то CNN показує кращі результати стиснення, ніж вони.

Це також забезпечує покращену продуктивність надвисокої роздільної здатності та зменшення артефактів. На додаток до цього, алгоритми стиснення даних на основі CNN покращують якість зображень JPEG. Це досягається шляхом зменшення пікового відношення сигнал/шум і структурної подібності.

Стиснення на основі CNN також може поєднуватися з продуктивністю стандарту High-Efficiency Video Coding. Це робиться за допомогою оцінки ентропії. Це також може бути дуже корисним для виконання стиснення файлів.

Стиснення на основі Generative Adversarial Network (GAN)

GAN можна визначити як альтернативу нейронним мережам, які використовують дві конкуруючі мережі. Це робиться для отримання більш точних аналізів і прогнозів. У 2017 році вперше були розроблені алгоритми на основі

GAN. Ці алгоритми стиснення даних можуть стискати файли більш ніж у два з половиною рази менше порівняно з традиційними широко використовуваними методами, такими як JPEG або WebP.

Алгоритми на основі GAN можна використовувати для стиснення в реальному часі з паралельним використанням обробки разом. Цей алгоритм працює, оскільки повністю стискає зображення на основі найбільш відповідних характеристик.

Коли виконується декодування, на основі прогнозів, які були зроблені цими ознаками, реконструюються зображення. Якщо порівняти його зі стисненням на основі CNN, стиснення на основі GAN створить для вас зображення дуже високої якості завдяки усуненню суперечливих втрат.

Прогноз за частковим збігом (PPM)

PPM – це адаптивний метод стиснення статистичних даних, заснований на контекстному моделюванні та передбаченні. Ці моделі використовують набір попередніх символів у нестисненому потоці символів, щоб передбачити наступний символ у потоці. Алгоритми PPM також можна використовувати для кластеризації даних у передбачені групи під час кластерного аналізу.

Кількість попередніх символів, n, визначає порядок моделі PPM, який позначається як PPM(n).

Необмежені варіанти, де контекст не має обмежень по довжині, також існують і позначаються як PPM. Якщо неможливо зробити прогноз на основі всіх n символів контексту, робиться спроба прогнозу з n − 1 символом. Цей процес повторюється, доки не буде знайдено відповідність або поки в контексті не залишиться символів. У цей момент робиться фіксований прогноз. Реалізації стиснення PPM значно відрізняються в інших деталях. Фактичний вибір символу зазвичай записується за допомогою арифметичного кодування, хоча також можна використовувати кодування Хаффмана або навіть певний тип техніки словникового кодування.

Кодування довжини серії (RLE)

RLW – це форма стиснення даних без втрат, у якій цикли даних (послідовності, у яких одне й те саме значення даних зустрічається в багатьох послідовних елементах даних) зберігаються як одне значення даних і підраховуються, а не як вихідний цикл. Це найбільш корисно для даних, які містять багато таких прогонів.

Наприклад, прості графічні зображення, такі як піктограми, штрихові малюнки, Гра життя Конвея та анімація. Це не корисно для файлів, які не мають багато прогонів, оскільки це може значно збільшити розмір файлу.

RLE також може використовуватися для позначення раннього формату графічного файлу, який підтримувався CompuServe для стиснення чорно-білих зображень, але був широко витіснений їх пізнішим форматом Graphics Interchange Format (GIF). RLE також відноситься до мало використовуваного формату зображення у Windows 3.x із правилом розширення, яке є закодованим растровим зображенням довжини серії, яке використовується для стиснення екрана запуску Windows 3.x.

bzip2

bzip2 – це безкоштовна програма для стиснення даних із відкритим кодом, яка використовує алгоритм Берроуза-Вілера. Він стискає лише окремі файли і не є архіватором файлів. Він розроблений Джуліаном Сьюардом і підтримується Федеріко Мена. Сьюард випустив перший публічний випуск bzip2, версія 0.15, у липні 1996 року. Стабільність і популярність компресора зростали протягом наступних кількох років, і Сьюард випустив версію 1.0 наприкінці 2000 року.

Після дев’ятирічної перерви в оновленнях проекту з 2010 року. 4 червня 2019 року Федеріко Мена прийняв підтримку проекту bzip2. bzip2 стискає дані в блоки розміром від 100 до 900 КБ.

Він використовує перетворення Берроуза-Вілера для перетворення послідовностей символів, що часто повторюються, у рядки ідентичних літер. Потім він застосовує перетворення переміщення вперед і кодування Хаффмана. Предок

bzip2 bzip використовував арифметичне кодування замість Хаффмана. Зміна була внесена через обмеження патенту на програмне забезпечення.

Кодування Хаффмана

Код Хаффмана – це особливий тип оптимального префіксного коду, який зазвичай використовується для стиснення даних без втрат. Процес пошуку або використання такого коду продовжується з використанням кодування Хаффмана, алгоритму, розробленого Девідом А. Хаффманом, коли він був доктором наук. студент Массачусетського технологічного інституту та опублікований у статті 1952 року «Метод побудови кодів з мінімальною надмірністю».

Вихідні дані алгоритму Хаффмана можна розглядати як кодову таблицю змінної довжини для кодування вихідного символу (наприклад, символу у файлі). Алгоритм отримує цю таблицю з оціненої ймовірності або частоти появи (ваги) для кожного можливого значення вихідного символу.

Як і в інших методах ентропійного кодування, більш поширені символи, як правило, представлені за допомогою меншої кількості бітів, ніж менш поширені символи. Метод Хаффмана може бути ефективно реалізований. Знаходження коду за час, лінійний до кількості вхідних ваг, якщо ці ваги відсортовані.

ZСтандарт

Zstandard (або zstd) – це алгоритм стиснення даних без втрат, розроблений Янном Коллетом у Facebook. Zstd є еталонною реалізацією в C. Версія 1 цієї реалізації була випущена як безкоштовне програмне забезпечення 31 серпня 2016 року.

Zstandard був розроблений для забезпечення коефіцієнта стиснення, порівнянного з коефіцієнтом стиснення алгоритму DEFLATE (розробленого в 1991 році та використовуваного в оригінальних ZIP і gzip програми), але швидше, особливо для декомпресії. Він налаштовується з рівнями стиснення в діапазоні від мінус 5 (найшвидший) до 22 (найнижча швидкість стиснення, але найкращий коефіцієнт стиснення).

Zstd на максимальному рівні стиснення дає ступінь стиснення, близький до LZMA, LZAHM і PPM. Він працює краще, ніж LZA або bzip2. Zstandard досягає поточної межі Парето, оскільки він розпаковується швидше, ніж будь-який інший доступний на даний момент алгоритм із подібним або кращим коефіцієнтом стиснення

Словники можуть мати великий вплив на ступінь стиснення невеликих файлів, тому Zstandard може використовувати наданий користувачем словник стиснення. Він також пропонує навчальний режим, здатний генерувати словник із набору зразків.

WavPack

WavPack – безкоштовний аудіокодек с відкритим вихідним кодом для стиску аудіо без втрати якості.

WavPack-формат (розширення. WV) дозволяє стискати (і відновлювати) 8-8-, 16-, 24- і 32-бітні аудіофайли в. WAV форматі. Він також підтримує багатоканальні потоки й високі частоти дискретизації (sampling rate). Як в інших способів компресії без втрати якості, ефективність стиску залежить від вихідних даних, але звичайно вона лежить у діапазоні між 30 % і 70 % для звичайної популярної музики, трохи вище для класичної музики й інших джерел з більше широким динамічним діапазоном.

WavPack також включає унікальний «гібридний» режим, що надає всі переваги стиску без втрат з додатковим бонусом: замість створення одного файлу, у цьому режимі створюється щодо невеликий файл високого (точніше, зазначеного при кодуванні) якості із втратою (.WV), що може програватися сам по собі, а також файл «корекції» (.WVC), що (у комбінації з попереднім .WV) дозволяє повністю відновити оригінал. Для деяких користувачів це означає, що їм ніколи не прийдеться вибирати між стиском без втрат і із втратою якості.

Висновок

Ці алгоритми стиснення даних допоможуть вам оптимізувати розмір файлу. Різні типи алгоритмів стиснення даних забезпечать різні результати. Однак,

якщо ви не можете знайти тут потрібний алгоритм, ви можете переглянути цей посібник і уточнити пошук. Алгоритмів не бракує, але ви повинні бути конкретними, коли шукаєте правильний алгоритм для свого проекту.

3.2 Розробка структурної схеми

На рисунку 3.1 зображена структурна схема розробленої системи. Структурна схема показує як блоки програми зв’язані між собою.

З нього ми бачимо, що система складається з наступних основних структурних блоків:

– Блок інтерфейсу користувача програми.

– Блок алгоритмів за допомогою яких відбувається архівування/розархівування файлів.

– Блок форматів архівів, з якими працює програма.

– Блок швидкого доступу до елементів програми.

– Блок виконання операцій.

Блок виконання операцій, у свою чергу, включає у себе наступні блоки:

– Архівування.

– Створення/розпакування багатотомних архівів.

– Розархівування.

– Відновлення пошкоджених архівів.

– Створення архівів, що саморозпаковуються.

Блок алгоритмів за допомогою яких відбувається архівування/розархівування файлів включає в себе реалізацію наступних алгоритмів:

– Алгоритми стиску без втрат.

– Алгоритми стиску із втратами.

– Алгоритми стиску з використанням словника.

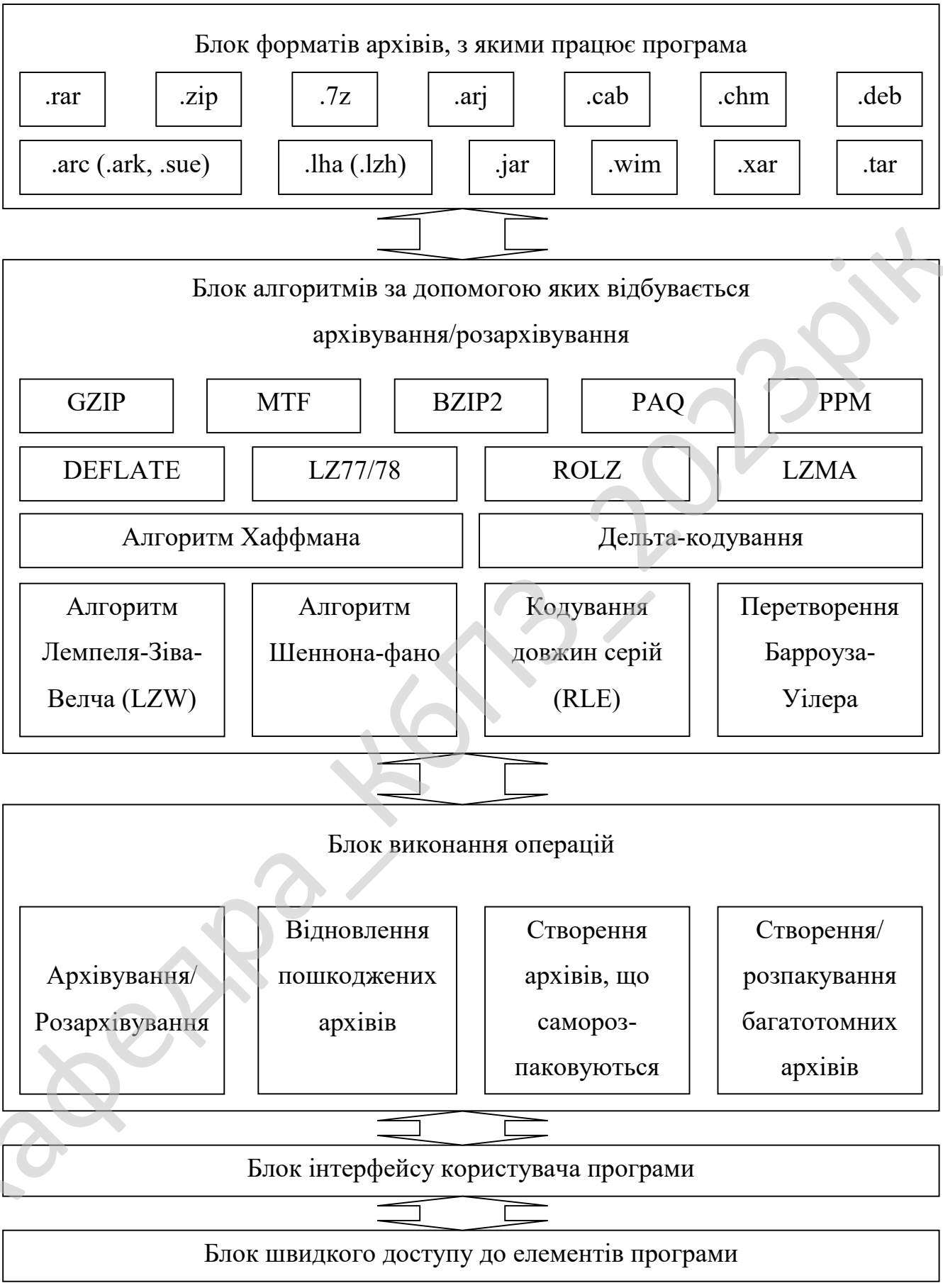

Рисунок 3.1 – Структурна схема системи

При цьому в архівуванні використовуються тільки алгоритми стиску без втрат.

Алгоритми стиску без втрат підрозділяються на наступні:

– gzip (скорочення від GNU Zip) – утиліта стиску й відновлення (декомпресії) файлів, що використовує алгоритм DEFLATE. Використовується в основному в UNIX-системах, у ряді яких є стандартом де-факто для стиску даних.

– LZ77 і LZ78 – алгоритми стиску без втрат, опубліковані в статтях Абрахама Лемпеля і Якоба Зіва в 1977 і 1978 роках. Ці алгоритми найбільш відомі варіанти в сімействі LZ*, що містить у собі також LZW, LZSS, LZMA і інші алгоритми. Обидва алгоритми відносяться до словникових методів, на відміну від інших методів зменшення надмірності, таких як RLE і арифметичний стиск. LZ77 є алгоритмом з «ковзним вікном», що еквівалентно неявному використанню словникового підходу, уперше запропонованого в LZ78.

– LZMA (Lempel-Ziv-Markov chain-Algorithm) – алгоритм стиску даних, розроблювальний з 2001 року, використовується в архіваторі 7-Zip для створення стислих архівів у форматі 7z. Алгоритм заснований на схемі стиску даних по словнику, подібній з використаною в LZ77, і забезпечує високий коефіцієнт стиску (звичайно перевищуючий коефіцієнт, одержуваний при стиску з використанням bzip2), а також дозволяє використовувати словники різного розміру (до 4 Гб).

– MTF – Рух до початку (move-to-front, MTF) – перетворення для кодування даних (звичайно потоку байтів) розроблене для поліпшення продуктивності ентропійного кодування. При гарній реалізації, воно достатньо швидко для включення як додатковий крок в алгоритмах стиску даних. Також може використовуватися разом з BWT, перетворенням Барроуза-Уілера.

– bzip2 – безкоштовна вільна утиліта командного рядка (а також алгоритм) з відкритим вихідним кодом для стиску даних. Розроблено й уперше опублікована Джуліаном Сьюардом у липні 1996 (версія 0.15). Стабільність і

популярність компресора росли протягом декількох років, і версія 1.0 була опублікована наприкінці 2000 року.

– Deflate – це алгоритм стиску без втрат, що використовує комбінацію алгоритму LZ77 і алгоритму Хаффмана. Він був описаний Філом Кацом для 2-ї версії своєї утиліти для створення архівів PKZIP, що згодом був визначений в RFC 1951. Deflate вважається вільним від всіх існуючих патентів, і поки патент на LZW (який використовується у форматі GIF) залишався в силі, це привело до використання deflate у файлах, стисливих gzip, і зображеннях у форматі PNG вдобавок до формату ZIP, для якого Кац його спроектував.

– PAQ – серія вільних архіваторів з текстовим інтерфейсом, які спільними зусиллями розроблювачів піднялися в перші місця рейтингів багатьох тестів стиску даних (хоча й ціною процесорного часу й обсягу пам'яті). Кращий результат у цій серії на більшості тестів був отриманий архіватором PAQ8JD, створеним спільними зусиллями Метта Махони, Олександра Ратушняка, Сергія Оснача, Пшемислава Скибиньского й Білла Петтиса, і випущеним 30 грудня 2006 року. Однак, у деяких тестах він відстає від WinRK (створеного Малькомом Тейлором у січні 2005 року) у режимі PWCM. PWCM (PAQ weighted context mixing, «PAQ зважене змішування контекстів») – стороння пропрієтарна реалізація алгоритму PAQ. Спеціально настроєні версії алгоритму PAQ виграли призи в Приз Хаттера й Калгари Корпус Челлендж.

– ROLZ (Reduced Offset LZ – алгоритм Лемпела-Зива зі скороченими зсувами) – словниковий алгоритм стиску даних, близький до LZ77, але використовуючий деякі контекстні прийоми для зменшення числа активних зсувів. Саме поняття ROLZ уперше ввів Malcolm Taylor у своєму архіваторі RK в 1999 році й даний алгоритм є одним з найбільш сучасних підходів до побудови швидких ефективних алгоритмів стиску.

– Алгори́тм Ле́мпеля-Зіва-Ве́лча (Lempel-Ziv-Welch, LZW) – це універсальний алгоритм стиску даних без втрат, створений Абрахамом Лемпелем (Abraham Lempel), Якобом Зівом (Jacob Ziv) і Терри Велчем (Terry Welch). Він

був опублікований Велчем в 1984 році, як поліпшена реалізація алгоритму LZ78, опублікованого Лемпелем і Зівом в 1978 році. Алгоритм розроблений так, щоб його можна було швидко реалізувати, але він не обов'язково оптимальний, оскільки він не проводить ніякого аналізу вхідних даних.

– Алгоритм Шеннона-фано – один з перших алгоритмів стиску, що вперше сформулювали американські вчені Шеннон і Фано. Даний метод стиску має велику подібність із алгоритмом Хаффмана, що з'явився на кілька років пізніше. Алгоритм використовує коди змінної довжини: символ, що часто зустрічається, кодується кодом меншої довжини, що рідко зустрічається – кодом більшої довжини. Коди Шеннона-Фано префіксні, тобто, ніяке кодове слово не є префіксом будь-якого іншого. Ця властивість дозволяє однозначно декодувати будь-яку послідовність кодових слів.

– PPM (Prediction by Partial Matching – пророкування по частковому збігу) – адаптивний статистичний алгоритм стиску даних без втрат, заснований на контекстному моделюванні й пророкуванні. Модель PPM використовує контекст – безліч символів у незжатому потоці, що передують даному, щоб пророкувати значення символу на основі статистичних даних. Сама модель PPM лише пророкує значення символу, безпосередній стиск здійснюється алгоритмами ентропійного кодування, як наприклад, алгоритм Хаффмана, арифметичне кодування. Довжина контексту, що використовується при пророкуванні звичайно сильно обмежена. Ця довжина позначається n і визначає порядок моделі PPM, що позначається як PPM(n). Необмежені моделі так само існують і позначаються просто PPM*. Якщо пророкування символу по контексту з n символів не може бути зроблено, то відбувається спроба пророчити його за допомогою n-1 символів. Рекурсивний перехід до моделей з меншим порядком відбувається поки пророкування не відбудеться в одній з моделей, або коли контекст стане нульової довжини (n=0). Моделі ступеня 0 і -1 варто описати особливо. Модель нульового порядку еквівалента случаю контекстно-вільного моделювання, коли імовірність символу визначається винятково із частоти його

появи в стисливому потоці даних. Подібна модель звичайно застосовується разом з кодуванням по Хаффману. Модель порядку -1 представляє собою статичну модель, яка присвоює імовірності символу визначене фіксоване значення; звичайно всі символи, які можуть зустрітися в стисливому потоці даних, при цьому вважаються рівноімовірними. Для одержання доброї оцінки імовірностей символу необхідно враховувати контексти різних довжин. PPM представляє собою варіант стратегії перемішування, коли оцінки імовірностей, зроблені на підставі контекстів різних довжин, поєднуються в одну загальну імовірність. Отримана оцінка кодується будь-яким ентропійним кодером (ЕК), звичайно це якийсь різновид аріфметичного кодера. На етапі ентропійного кодування й відбувається властиво стиск. Велике значення для алгоритму PPM має проблема обробки нових символів, що ще не зустрічалися у вхідному потоці. Це проблема зветься проблема нульової частоти. Деякі варіанти реалізацій PPM думають лічильник нового символу рівним фіксованій величині, наприклад, одиниці. Інші реалізації, як наприклад, PPM-D, збільшують псевдолічильник нового символу щораз, коли, дійсно, у потоці з'являється новий символ. (Інакше кажучи, PPM-D оцінює ймовірність появи нового символу як відношення числа унікальних символів до загального числа використовуваних символів). Опубліковані дослідження алгоритмів сімейства PPM з'явилися в середині 1980-х років. Програмні реалізації не були популярні до 1990-х років, тому як моделі PPM вимагають значну кількість оперативної пам'яті. Сучасні реалізації PPM є кращими серед алгоритмів стиску без втрат для текстів природною мовою.

– Дельта-кодування (Delta encoding) – спосіб збереження або передачі даних у формі різниці (дельти) між послідовними даними замість самих даних. Це часто називається дельта-компресія, тому що деякі зразки кодування можуть одержувати кодовані дані в більше короткому виді, чим вихідні дані.

– Алгоритм Хаффмана – адаптивний жадібний алгоритм оптимального префіксного кодування алфавіту з мінімальною надмірністю. Був розроблений в 1952 році аспірантом Массачусетского технологічного інституту Девідом

Хаффманом при написанні їм курсової роботи. У цей час використовується в багатьох програмах стиску даних. На відміну від алгоритму Шеннона-фано, алгоритм Хаффмана залишається завжди оптимальним і для вторинних алфавітів m_2 з більш ніж двома символами.

– Кодування довжин серій (Run-length encoding, RLE) або Кодування повторів – простий алгоритм стиску даних, що оперує серіями даних, тобто послідовностями, у яких той самий символ зустрічається кілька разів підряд. При кодуванні рядок однакових символів, що становлять серію, заміняється рядком, що містить сам повторюваний символ і кількість його повторів.

– Перетворення Барроуза-Уілера (Burrows-Wheeler transform, BWT, також історично називається стиском, що блочно-сортує, хоча стиском і не є) – це алгоритм, використовуваний у техніках стиску даних для перетворення вихідних даних. BWT використовується в архіваторі bzip2. Алгоритм був винайдений Майклом Барроузом і Девідом Уілером. Терміном BWT також називають і повні алгоритми стиску, що використовують BWT як один із кроків.

Блок форматів архівів, з якими працює програма, включає в себе наступні формати:

– RAR – розповсюджений проприетарний формат стиску даних і програма-архіватор. Формат розроблений російським програмістом Євгенієм Рошалом (звідси й назва RAR: Roshal Archiver). Він написав програму-архіватор для впакування/розпакування RAR, під DOS, потім і для інших операційних систем. Версія для Microsoft Windows поширюється в складі багатоформатного архіватора із графічним інтерфейсом WinRAR. Програма поширюється як умовно-безкоштовне програмне забезпечення (shareware). Для файлів формату RAR звичайно використовується розширення .rar і MIME-Тип application/ x-rar-compressed.

– ZIP – популярний формат стиску даних і архівування файлів. Файл у цьому форматі звичайно має розширення .zip і зберігає в стислому або незжатому виді один або кілька файлів, які можна з нього витягти шляхом розпакування за

допомогою спеціальної програми. ZIP був розроблений Філом Кацем для використання в програмі PKZIP. Згодом з'явилася безліч інших утиліт, що працюють із цим форматом.

– 7z – формат алгоритму 7-Zip – вільний файловий архіватор з високим ступенем стиску даних. Підтримує кілька алгоритмів стиску й безліч форматів даних, включаючи власний формат 7z с високоефективним алгоритмом стиску LZMA. Програма розробляється з 1999 року і є безкоштовною, а також має відкритий вихідний код, більша частина якого вільно поширюється на умовах ліцензії GNU LGPL, за винятком коду декомпресора unRAR, що має обмеження. Основною платформою є Windows, де доступні дві версії програми: із графічним інтерфейсом і версія для командного рядка. Консольна версія була портирована співтовариством розроблювачів для систем стандарту POSIX під загальною назвою p7zip. Портировані версії для інших систем, так само як і оригінальна програма 7-zip, доступні на сайті системи SourceForge. 7-Zip є переможцем SourceForge.net Community Choice Awards 2007 року в категоріях: кращий проект і кращий технічний дизайн.

– ARJ – файловий архіватор. Розроблений Робертом К. Джангом (Robert K. Jung). (Походження найменування ARJ: Archiver Robert Jung). ARJ компресія подібна PKZIP 1.02. Існує також версія ARJ з відкритим вихідним кодом, доступна під більш, ніж десятьома операційними системами, включаючи DOS, 16- і 32-х розрядні версії Windows і OS/2, різні варіанти UNIX і Linux. Існує також версія Russian NVL, що дозволяє захищати архіви за допомогою шифрування алгоритмом GOST.

– ARC – формат стиску даних без втрат і архівування файлів від System Enhancement Associates. Файл у цьому форматі звичайно має розширення .arc, .ark або .sue і зберігає в стислому або незжатому виді один або кілька файлів, які можна з нього витягти шляхом розпакування за допомогою спеціальної програми.

– Cabinet (.cab) – формат файлів для архівів зі стиском, що застосовується в операційних системах сімейства Microsoft Windows. Формат підтримує стиск і

цифрові підписи, використовується в різних технологіях установників від Microsoft: Setup API, Device Installer, AdvPack (для установки компонентів Active через Internet Explorer) і Windows Installer.

– HTMLHelp (Microsoft Compressed HTML Help, Microsoft Compiled HTML Help, .CHM) – пропріетарний формат файлів контекстної довідки, розроблений корпорацією Microsoft і випущений в 1997 році як заміна формату WinHelp. Містить у собі набір HTML-сторінок, може також містити в собі зміст із посиланнями на сторінки, предметний покажчик, а також базу для повнотекстового пошуку по вмісту сторінок. Всі вхідні в.CHM файли стислі алгоритмом LZX. Для перегляду .CHM-файлів використовується стандартний засіб перегляду, убудоване в усі версії Microsoft Windows, починаючи з Windows 98, і Windows NT. Крім того, існує ряд сторонніх програм-проглядачів, FBReader, etc. Для створення .CHM-файлів «Майкрософт» надає безкоштовний засіб HTML Help Workshop.

– deb – розширення імен файлів «бінарних» пакетів для поширення й установки програмного забезпечення в ОС проекту Debian, і інших, що використовують систему керування пакетами dpkg.

– JAR файл – це Java-архів. Являє собою звичайний ZIP-архів, у якому втримується частина програми мовою Java. Щоб JAR файл був що виконується, він повинен містити файл MANIFEST.MF у каталозі META-INF, у якому повинен бути зазначений головний клас програми (такий клас повинен містити метод main). Номер версії JAR задається параметром Manifest-Version і є обов'язковим. В SDK 1.2 значення цього параметра повинне бути дорівнює 1.0.

– LHA – безкоштовний архіватор і відповідний формат архівування файлів (які мають розширення ім'я .LZH). Як і прабатько, розроблявся для архівування текстових файлів.

– Windows Imaging Format (WIM) – це файл-орієнтований формат образа диска. Формат був розроблений компанією Microsoft для розгортання останніх релізів операційних систем сімейства Windows – Windows Vista/7 і Windows

Server 2008, які використовують його як частину стандартної процедури установки. Втім, його можна використовувати й з іншими релізами Windows; крім того він застосовується в Windows Fundamentals for Legacy PCs – компактної ОС від Microsoft для застарілих PC, створеної на базі Microsoft Windows XP Embedded Service Pack 2 і вийшла 8 липня 2006 року.

– xar (eXtensible ARchive format) – це формат стиску даних і архівування файлів з відкритим вихідним кодом. Файл у цьому форматі звичайно має розширення .xar і зберігає в стислому або незжатому виді один або кілька файлів, які можна з нього витягти шляхом розпакування за допомогою спеціальної програми. XAR був створений у рамках проекту OpenDarwin і використовується в операційній системі Mac OS X 10.5 для процедури установки програмного забезпечення.

– tar – формат бітового потоку або файлу архіву, а також назва традиційної для Unix програми для роботи з такими архівами. Програма tar була стандартизована в POSIX.1-1998, а також пізніше в POSIX.1-2001. Спочатку програма tar використовувалася для створення архівів на магнітній стрічці, а в наш час tar використовується для зберігання декількох файлів усередині одного файлу, для поширення програмного забезпечення, а також по прямому призначенню – для створення архіву файлової системи. Одним з переваг формату tar при створенні архівів є те, що в архів записується інформація про структуру каталогів, про власника й групу окремих файлів, а також тимчасові мітки файлів.

3.3 Розробка функціональної схеми

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

Функціональна схема складається з наступних блоків:

– Блок інтерфейсу користувача програми.

– Блок швидкого доступу до елементів програми.

– Блок роботи з файлами.

– Блок виконання команд.

– Блок операцій.

– Блок параметрів.

– Довідка.

Блок швидкого доступу до елементів програми включає до себе наступні елементи:

– Додати в архів.

– Витягти з архіву.

– Тест.

– Перегляд архіву.

– Видалення.

– Знайти.

– Майстер роботи з архівом.

– Інформація.

– Виправити помилки в архіві.

Блок роботи з файлами включає в себе:

– Відкрити архів.

– Вибрати диск.

– Встановити пароль.

– Скопіювати файли у буфер обміну.

– Вставити файли з буферу обміну.

– Виділити усе.

– Зняти виділення.

– Вихід.

Блок виконання команд включає в себе наступні команди:

– Додати файли у архів.

– Витягти у вказану папку.

– Протестувати файли у архіві.

– Переглянути файл.

– Видалити файл.

– Перейменувати файл.

– Файл на друк.

– Додати архівний коментар.

– Додати інформацію для відновлення.

– Заблокувати архів.

Блок операцій включає у себе наступні операції:

– Майстер архіву.

– Перевірити архіви на віруси.

– Перетворити архіви.

– Відновити архіви.

– Перетворити архіви в SFX.

– Знайти файли.

– Показати інформацію.

– Створити звіт.

Блок параметрів включає в себе настроювання наступних параметрів:

– Настроювання.

– Імпорт/експорт файлу.

– Список файлів.

– Дерево файлів.

– Теми.

Довідка включає в себе наступні пункти:

– Зміст.

– Web-сторінка програми.

– Про програму.

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

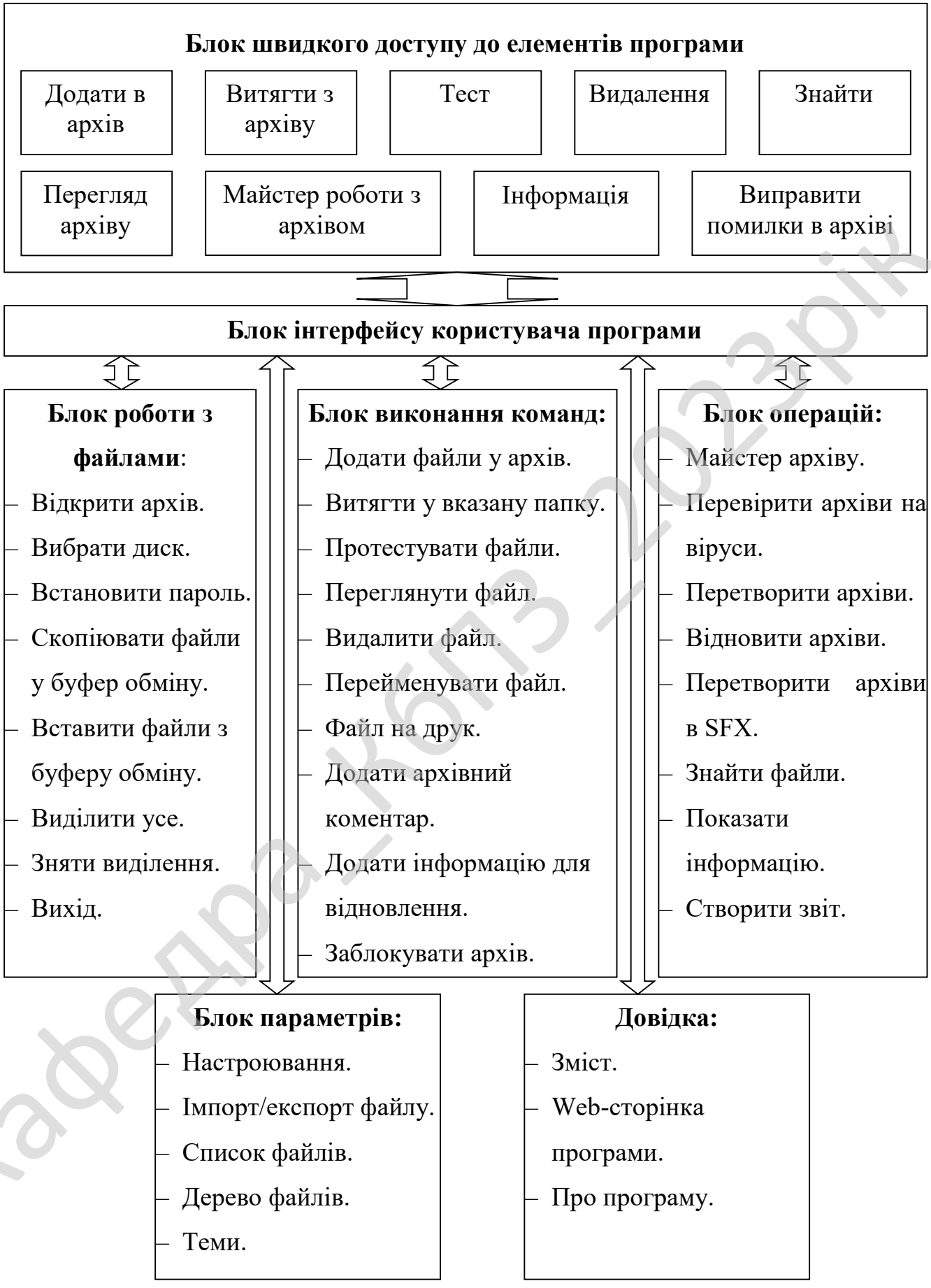

Рисунок 3.2 – Функціональна схема системи

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання бакалаврського проектування, наведена на рисунку 3.3.

У системі взаємодіють наступні процеси.

Першим завантажується процес початку роботи програмного забезпечення.

Він взаємодіє з процесом перегляду архівів.

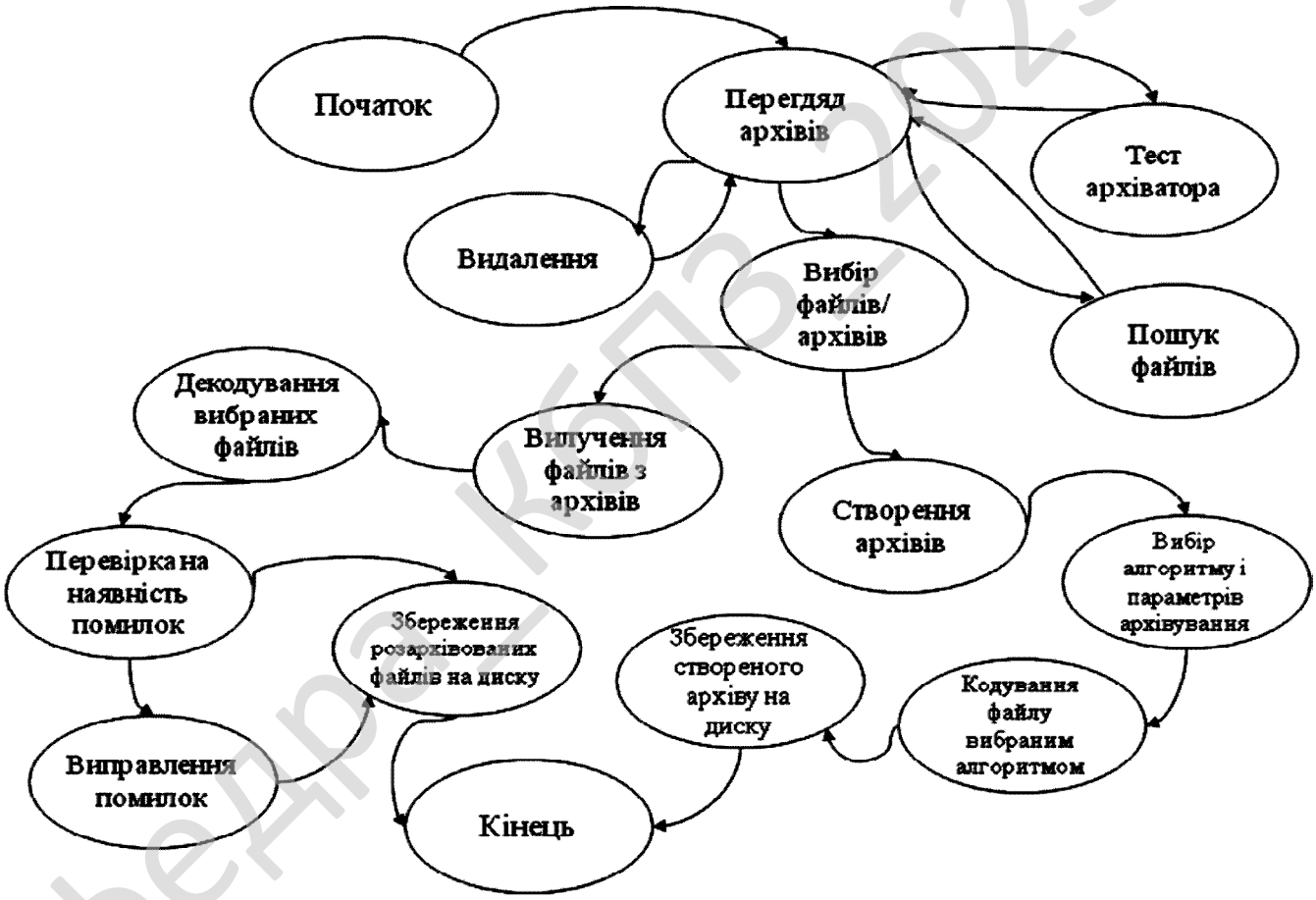

Рисунок 3.3 – Діаграма взаємодії процесів системи

Процес перегляду архівів взаємодіє з наступними процесами:

– Процес видалення.

– Процес вибору файлів/архівів.

– Процес тестування архіватора.

– Процес пошуку файлів.

Процес вибору файлів/архівів взаємодіє з наступними процесами:

– Процес створення архівів.

– Процес вилучення файлів з архівів.

Процес створення архівів взаємодіє з процесом вибору алгоритму та параметрів архівування.

Процес вибору алгоритму та параметрів архівування, у свою чергу, взаємодіє з процесом кодування файлу обраним алгоритмом.

Після цього запускається процес збереження створеного архіву на диску.

З іншого боку, процес вилучення файлів з архівів взаємодіє з процесом декодування вибраних файлів.

Процес декодування вибраних файлів взаємодіє з процесом перевірки на наявність помилок у розархівованому файлі.

Цей процес взаємодіє з наступними процесами:

– Процес виправлення помилок.

– Процес збереження розархівованих файлів на диску.

Ці процеси, є завершальними у системі.

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

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 наведено блок-схему основної програми. Її робота складається з виконання наступних кроків.

Спершу відбувається виведення основного вікна програми. Після цього відбувається перегляд архіву.

Наступним кроком є обрання файлів або архівів, в залежності від операції, яку над ними будуть проводити.

Якщо буде створюватись архів, тоді виконуються наступні операції:

– Вибір алгоритму та параметрів архівування.

– Кодування алгоритмів.

– Збереження архівів.

Якщо ж буде проводитися розархівування файлів, тоді будуть виконуватися наступні дії:

– Декодування вибраних файлів.

– Перевірка на наявність помилок.

Якщо виявлені помилки, то визначається, чи можливо помилки виправити.

Якщо їх можливо виправити, тоді помилки виправляються.

У іншому випадку виводиться повідомлення, про те, що помилки неможливо виправити, й вказується тип помилки.

Після цього відбувається збереження розархівованих файлів.

Далі користувач обирає, працювати йому з системою далі, або ні.

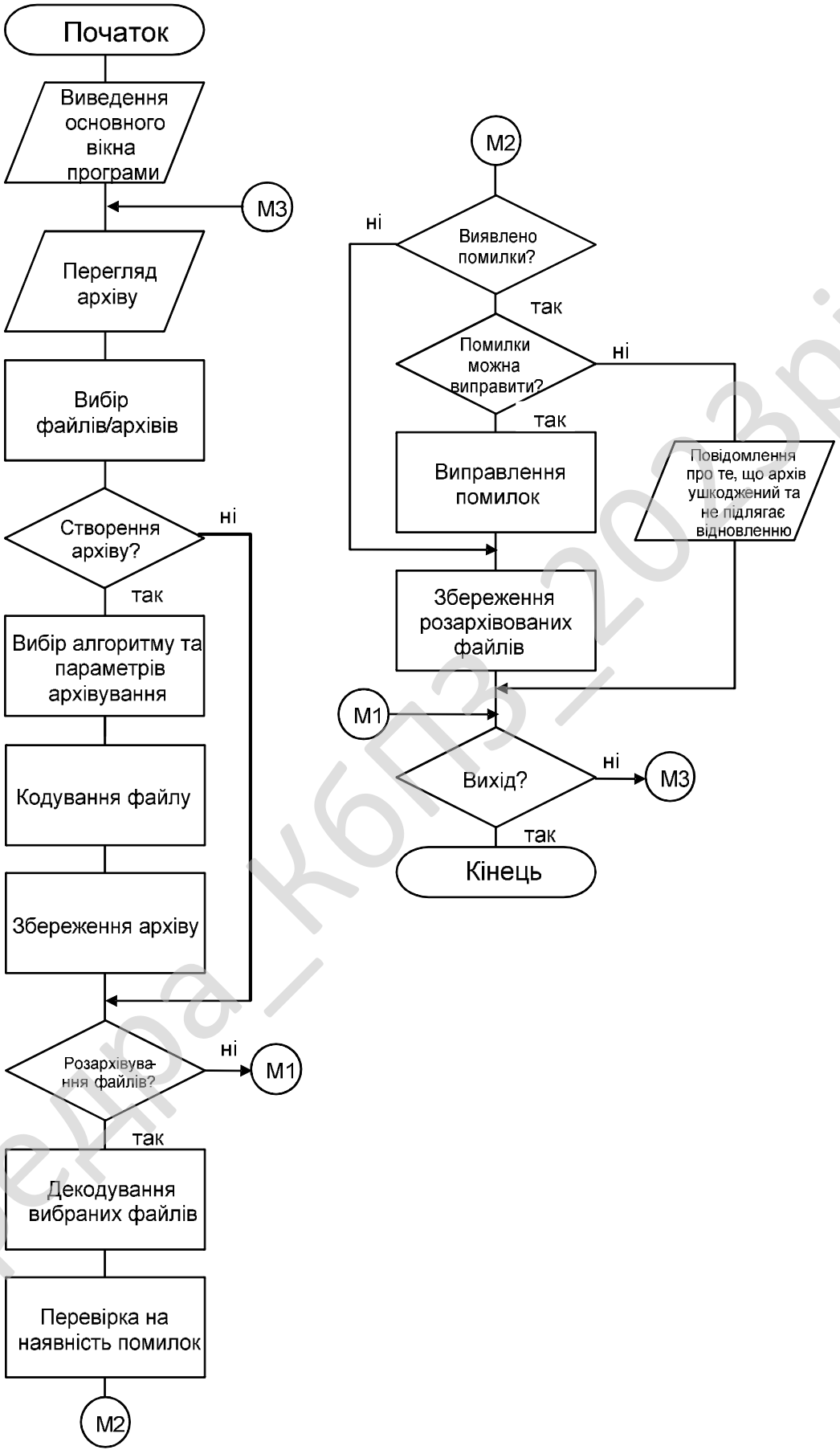

Рисунок 4.1 – Блок-схема роботи основної програми

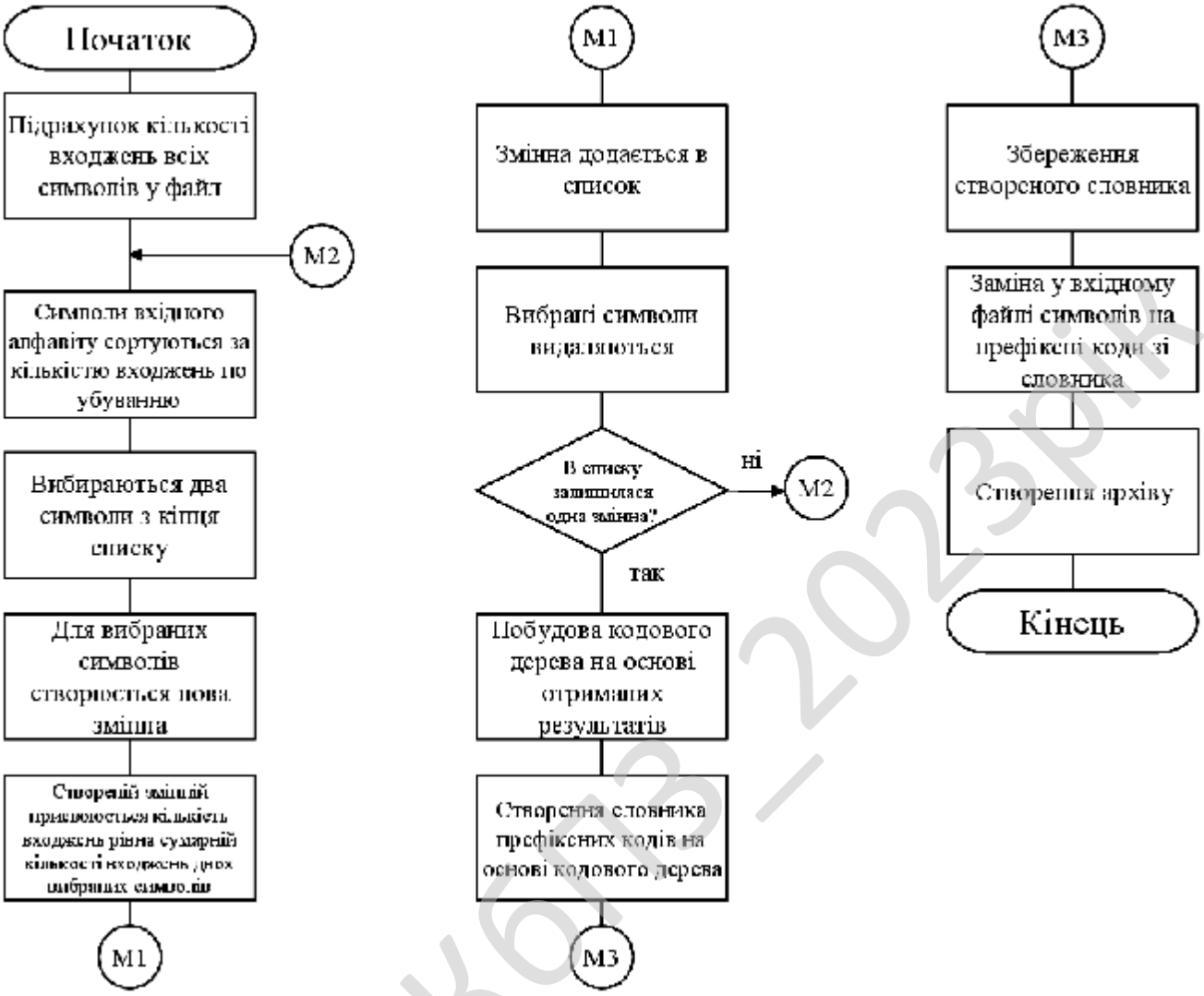

Рисунок 4.2 – Блок-схема підпрограми архівування методом Хаффмана

На рисунку 4.2 зображено блок-схема підпрограми архівування методом Хаффмана. З нього ми бачимо, що ця підпрограма працює наступним чином.

Спершу відбувається підрахунок кількості усіх входжень символів у файл.

Після цього символи вхідного алфавіту сортуються за кількістю входжень по убуванню.

На наступному етапі обираються два символи з кінця списку.

Для обраних символів створюється нова зміна.

Створеній змінній присвоюється кількість входжень, яка дорівнює сумарній кількості входжень двох обраних символів.

Змінна додається у список.

Обрані символи видаляються.

Після цього відбувається перевірка на те скільки змінних осталося в списку.

Якщо змінна залишилася одна, тоді виконуються наступні дії:

– Будується кодове дерево на основі отриманих результатів.

– Створюється словник префікс них кодів на основі кодового дерева.

– Відбувається збереження створеного словника.

– У вхідному файлі відбувається заміна символів на префіксні коди зі словника.

– Створюється архів.

Якщо ж змінних залишилося декілька, тоді повертаємось на етап обирання двох символів з кінця списку.

На цьому піпрограмма завершує свою працю.

Наведемо частину коду, яка реалізує цю підпрограму.

```
//Процедура заміни двох останніх символів на один
procedure AddEl(var start: TPElSp; PNew: TPElSp);
var
 WP: TPElSp;
begin
 PNew^.next:= nil;
 if start = nil then start:= PNew
 else
  begin
  WP:= start;
  while WP^.next <> nil do
   WP:= WP^.next;
  WP^.next:= PNew;
end;
end;
//Функція пошуку куди вставити замінений символ
function FindEl(start: TPElSp; key: byte; var FindPoint: TPElSp): boolean;
begin
 if start = nil then
 begin
  result:= false;
  exit;
```

```
 end;
 result:= false;
 FindPoint:= start;
 while (FindPoint <> nil) and (FindPoint^.key <> key) do
  begin
  findpoint:= findpoint^.next;
  end;
  if (findpoint <> nil) then result:= true;
end;
//Функція визначення скільки елементів осталось у рядку
function FindElMax(start: TPElSp; var FindPoint: TPElSp): boolean;
var
 PrevPoint: TPElSp;
begin
 if start = nil then
 begin
  result:= false;
  exit;
 end;
 FindPoint:= start;
 PrevPoint:= start^.next;
 while (PrevPoint <> nil) do
  begin
  if PrevPoint^.cnt > FindPoint^.cnt then
   FindPoint:= PrevPoint;
  Prevpoint:= PrevPoint^.next;
  end;
  result:= true;
end;
//процедура виделення елементів, які були замінені
procedure DelEl(var start: TPElSp; key: byte);
var
PrevPoint, WPoint: TPElSp;
begin
if start = nil then exit;
 prevpoint:= nil;
 wpoint:= start;
 while (wpoint <> nil) and (wpoint^.key <> key) do
 begin
  prevpoint:= wpoint;
  wpoint:= wpoint^.next;
 end;
```

```
 if (wpoint = nil) or (wpoint^.key > ord(key)) then
 exit;
 if prevpoint = nil then
  start:= start^.next
 else
  prevpoint^.next:= wpoint^.next;
  Dispose(Wpoint);
end;
//Функція перетворення рядка у двійкове число
function StrBToInt(bin: string): byte;
var
i: integer; v: byte;
begin
  v:= 0;
  result:= 0;
  for i:=length(bin) downto 1 do
  begin
    result:= result + StrToInt(bin[i]) * StrToInt(FloatToStr(exp(ln(2)*v)));
    inc(v);
  end;
end;
//Функція перетворення двійкового числа у рядок
function IntToStr(in_int: byte): string;
var
 k: TKey;
 t1, t2: byte;
begin
  PBS:= 0;
  t1:= in_int;
  t2:= in_int;
  result:= '';
  while (t2 <> 1) and (t2 <> 0) do
  begin
   t1:= t2 mod 2;
   k:= IntToStr(t1);
    push(k);
    t2:= t2 div 2;
  end;
   k:= IntToStr(t2);
   push(k);
  while (PBS <> 0) do
  begin
```

```
    pop(k);
    result:= result + k;
   end;
end;
//Процедура кодування
procedure CreateKod(head: TPArNode; var ar: TArTree; size: byte);
var
 k: TKey;
begin
 if (head = nil) then exit;
   push(head^.kod);
   CreateKod(head^.left, ar, size);
   tmpi:= FindElAr(ar, size, head^.symb);
   cnti:=0;
   while (PBS<>0) do
   begin
    inc(cnti);
    pop(k);
    ar[tmpi].huff:= ar[tmpi].huff + k;
   end;
   reverse(ar[tmpi].huff);
   PBS:= PBS + cnti;
   CreateKod(head^.right, ar, size);
   dec(PBS);
end;
//Функція закінчення кодування методом Хаффмана
function EndHaffman(a: TArTree; size: integer): extended;
var
 tmp: TPArNode;
begin
 tmp:= a[ size-1].st_tree;
 while (tmp^.parent <> nil) do
  tmp:= tmp^.parent;
 result:= tmp^.value;
end;
//Процедура оптимізації кода Хаффмана
procedure OptimizeKod(var ar: TArTree; size: byte);
var
 i, j: integer; ok: boolean; str: string; news, tmp: string[1];
begin
 for i:= 0 to size-1 do
 begin
```

```
  ok:= true;
  SetLength(str, length(ar[i].huff));
  str:= ar[i].huff;
  tmp:= str[1];
  if tmp = '0' then
  begin
   for j:=1 to length(str) do
    if (str[j] <> '0') then
     ok:= false;
   end else
     ok:= false;
   if ok then
   begin
   SetLength(news, length(ar[i].huff));
   for j:=1 to length(ar[i].huff) do
    news[j]:= '1';
    ar[i].huff:= news;
   end;
 end;
end;
//Функція читання тексту
function GetText(ar: TArTree; size: byte): widestring;
var
 i: byte;
begin
result:= '';
 for i:= 0 to size-1 do
  result:= result + a[i].huff;
end;
//Процедура підрахунку усіх елементів
procedure Countall(a: TArTree; size: byte);
var
 i: integer;
 H, Hmax, LiPi, Kss, Koe: real;
begin
  Hmax:= log2(size);
  H:= 0;
  LiPi:= 0;
  for i:= 0 to size - 1 do
  begin
   H:= H + a[i].n0 * log2(a[i].n0);
   LiPi:= LiPi + length(a[i].huff) * a[i].n0;
```

```
  end;
  H:= (-1)*H;
  liPi:= log2(LiPi);
  Kss:= Hmax / LiPi;
  Koe:= H / LiPi;
  Form1.StringGrid1.Cells[1,0]:= FloatToStr(Hmax);
  Form1.StringGrid1.Cells[1,1]:= FloatToStr(H);
  Form1.StringGrid1.Cells[1,2]:= FloatToStr(LiPi);
  Form1.StringGrid1.Cells[1,3]:= FloatToStr(Kss);
  Form1.StringGrid1.Cells[1,4]:= FloatToStr(Koe);
end;
//Функція заміни згідно словника
function GetFromKod(a: TArFRec; size: byte; key: string; var pos: byte):
boolean;
var
 i: byte;
begin
 result:= false;
 pos:= 0;
 for i:=0 to size-1 do
  if a[i].kod = key then
  begin
   result:= true;
   pos:= i;
   exit;
  end;
end;
//Функція перевірки на помилки
function CorrectKod(str: string; itSize: byte): string;
var
 i, j, k: integer; tmp: string[16];
begin
  if length(str) = itSize then
  begin
   result:= str;
   exit;
  end;
  tmp:= '0000000000000000';
  i:= itSize - length(str);
  delete(tmp, itSize+1, 16);
  k:= 1;
  for j:=i+1  to itSize do
```

```
    begin
      tmp[j]:= str[k];
      inc(k);
    end;
    result:= tmp;
end;
```

Формат архівного файлу RAR

Файл архіву складається із блоків різної довжини. Порядок проходження цих блоків може мінятися, але першим блоком завжди повинен бути блок-маркер, за яким треба блок заголовка архіву.

Кожний блок починається з наступних полів:

HEAD_CRC 2 байти CRC усього блоку або його частини.

HEAD_TYPE 1 байт Тип блоку.

HEAD_FLAGS 2 байти Прапори блоку.

HEAD_SIZE 2 байти Розмір блоку.

ADD_SIZE 4 байти Необов'язкове поле – додавання до розміру блоку.

Поле ADD_SIZE є присутнім, тільки якщо (HEAD_FLAGS & 0x8000) != 0.

Загальний розмір блоку зазначений у поле HEAD_SIZE, якщо (HEAD_FLAGS & 0x8000) == 0 або HEAD_SIZE+ADD_SIZE, якщо є поле ADD_SIZE, при цьому (HEAD_FLAGS & 0x8000) != 0.

У всіх блоках наступні біти в HEAD_FLAGS мають однакове значення:

– 0x4000 – якщо встановлено, те старі версії RAR будуть ігнорувати цей блок і видаляти його при зміні архіву; якщо не встановлений, то блок копіюється в новий архівний файл при зміні архіву;

– 0x8000 – якщо встановлено, то є присутнім поле ADD_SIZE, і розмір повного блоку становить HEAD_SIZE+ADD_SIZE.

Заявлені типи блоків:

HEAD_TYPE=0x72 блок-маркер

HEAD_TYPE=0x73 заголовок архіву

HEAD_TYPE=0x74 заголовок файлу

HEAD_TYPE=0x75 заголовок коментарю старого типу

HEAD_TYPE=0x76 електронний підпис старого типу

HEAD_TYPE=0x77 субблок старого типу

HEAD_TYPE=0x78 інформація для відновлення старого типу

HEAD_TYPE=0x79 електронний підпис старого типу

HEAD_TYPE=0x7a субблок

Блок коментарю використовується тільки усередині інших блоків.

Обробка архіву відбувається в такий спосіб:

1. Читається й перевіряється блок-маркер.

2. Читається заголовок архіву.

3. Читаються або пропускаються HEAD_ SIZE-Розмір(MAIN_HEAD) байт.

4. Якщо виявлено кінець архіву, то обробка архіву припиняється, інакше читаються 7 байт у полях HEAD_CRC, HEAD_TYPE, HEAD_FLAGS, HEAD_SIZE.

5. Перевіряється HEAD_TYPE.

Якщо HEAD_TYPE==0x74 прочитати заголовок файлу (перші 7 байт уже прочитані) прочитати або пропустити HEAD_ SIZE-Розмір(FILE_HEAD) байт

Якщо (HEAD_FLAGS & 0x100) прочитати або пропустити HIGH_PACK_SIZE*0x100000000+PACK_SIZE байт інакше прочитати або пропустити FILE_SIZE байт інакше прочитати відповідний блок HEAD_TYPE: прочитати HEAD_ SIZE-7 байт якщо (HEAD_FLAGS & 0x8000) прочитати ADD_SIZE байт

6. Перейти до кроку 4.

Формати блоків.

Блок-маркер (MARK_HEAD):

HEAD_CRC Завжди 0x6152.

2 байти

HEAD_TYPE Тип заголовку: 0x72.

1 байт

HEAD_FLAGS Завжди 0x1a21.

2 байти

HEAD_SIZE Розмір блоку = 0x0007.

2 байти

Блок-маркер у дійсності вважається фіксованою послідовністю байт: 0x52 0x61 0x72 0x21 0x1a 0x07 0x00.

Заголовок архіву (MAIN_HEAD):

HEAD_CRC CRC полів від HEAD_TYPE до RESERVED2

2 байти

HEAD_TYPE Тип заголовка: 0x73

1 байт

HEAD_FLAGS Бітові прапори:

2 байти:

0x0001 – Атрибут тому (том багатотомного архіву).

0x0002 – Є присутнім архівний коментар
RAR 3.x використовує окремий блок коментарю
і не встановлює цей прапор.

0x0004 – Атрибут блокування архіву.

0x0008 – Атрибут безперервного (solid) архіву.

0x0010 – Нова схема іменування томів ('volname.part.rar').

0x0020 – Є присутнім інформація про автора або
електронний підпис (AV).
RAR 3.x не встановлює цей прапор.

0x0040 – Є присутнім інформація для відновлення.

0x0080 – Заголовки блоків зашифровані.

0x0100 – Перший том (установлює тільки RAR 3.0 і старше).

Інші біти в HEAD_FLAGS зарезервовані для
внутрішнього використання.

HEAD_SIZE Загальний розмір архівного заголовка, включаючи архівні

2 байти коментарі.

RESERVED1 Зарезервовано.

2 байти

RESERVED2 Зарезервовано.

4 байти

Заголовок файлу (файл в архіві):

HEAD_CRC CRC полів від HEAD_TYPE до FILEATTR і ім'я файлу.

2 байти

HEAD_TYPE Тип заголовка: 0x74.

1 байт

HEAD_FLAGS Бітові прапори:

2 байти

0x01 – файл триває з попереднього тому.

0x02 – файл триває в наступному томі.

0x04 – файл зашифрований паролем.

0x08 – є присутнім коментар файлу.

RAR 3.x використовує окремий блок коментарю і не встановлює цей прапор.

0x10 – використовується інформація з попередніх файлів (прапор безперервності) (для RAR 2.0 і старше).

біти 7 6 5 (для RAR 2.0 і вище):

0 0 0 – розмір словника 64 Кб.

0 0 1 – розмір словника 128 Кб.

0 1 0 – розмір словника 256 Кб.

0 1 1 – розмір словника 512 Кб.

1 0 0 – розмір словника 1024 Кб.

1 0 1 – розмір словника 2048 Кб.

1 1 0 – розмір словника 4096 Кб.

1 1 1 – файл є каталогом.

0x100 – присутні поля HIGH_PACK_SIZE і HIGH_UNP_SIZE. Ці поля використовуються тільки для архівування дуже більших файлів (більше 2 Гб), для файлів меншого обсягу ці поля відсутні.

0x200 – FILE_NAME містить імена у звичайному форматі й в Unicode, розділені нулем. У цьому випадку поле NAME_SIZE дорівнює довжині звичайного ім'я плюс довжина ім'я у форматі Unicode плюс 1.
Якщо цей прапор присутній, а FILE_NAME не містить нульових байт, це означає, що ім'я файлу закодоване у форматі UTF-8.

0x400 – після ім'я файлу в заголовку перебуває 8 додаткових байт, які необхідні для збільшення надійності шифрування (так звана "сіль").

0x800 – прапор версії. Це стара версія файлу, номер версії доданий до ім'я файлу як ';n'.

0x1000 – є присутнім поле розширеного часу.

0x8000 – цей біт завжди встановлений, тому що загальний розмір блоку HEAD_SIZE + PACK_SIZE (і плюс HIGH_PACK_SIZE, якщо встановлено біт 0x100).

HEAD_SIZE 2 байти — Повний розмір заголовка файлу, включаючи ім'я файлу й коментарі.

PACK_SIZE 4 байти — Розмір файлу в архіві (стислий).

UNP_SIZE 4 байти — Розмір вихідного файлу (незжатий).

HOST_OS 1 байт — Використана при архівуванні операційна система: 0 – MS-DOS.

1 – OS/2.

2 – Win32.

3 – Unix.

4 – Mac OS .

5 – BeOS.

FILE_CRC CRC файлу.

4 байти

FTIME Дата й час у стандартному форматі MS-DOS.

4 байти

UNP_VER Версія RAR, необхідна для добування файлу.

1 байт Номер версії кодується як

10 * старший номер версії + молодший номер версії.

METHOD Метод стиску:

1 байт 0x30 – збереження без стиску.

0x31 – швидкісний стиск.

0x32 – швидкий стиск.

0x33 – звичайний стиск.

0x34 – гарний стиск.

0x35 – максимальний стиск.

NAME_SIZE Розмір ім'я файлу.

2 байти

ATTR Атрибути файлу.

4 байти

HIGH_PACK_SIZE Старші 4 байти 64-бітового значення

4 байти розміру стислого файлу.

Необов'язкове значення, що є присутнім, тільки якщо установлено біт 0x100 в HEAD_FLAGS.

HIGH_UNP_SIZE Старші 4 байти 64-бітового значення

4 байти розміру незжатого файлу.

Необов'язкове значення, що є присутнім, тільки якщо установлено біт 0x100 в HEAD_FLAGS.

FILE_NAME Ім'я файлу – рядок розміром NAME_SIZE байт.

SALT Є присутнім, якщо (HEAD_FLAGS & 0x400) != 0.

8 байт

EXT_TIME Є присутнім, якщо (HEAD_FLAGS & 0x1000) != 0.

Змінний розмір.

Тут можуть бути інші нові поля.

Примітки:

1. Для обробки SFX-архіву потрібно пропустити модуль SFX і знайти в архіві блок-маркер. У самому SFX-модулі послідовність байтів блоку-маркера (0x52 0x61 0x72 0x21 0x1a 0x07 0x00) відсутня.

2. CRC обчислюється за допомогою стандартного полінома 0xEDB88320. У випадку якщо розмір CRC менше 4 байт, використовуються тільки молодші байти.

Створення zip-архівів і добування з них інформації

Для рішення даного завдання будемо користуватися компонентом (вірніше, цілою колекцією компонентів) за назвою ZipTV. Я вибрав його тому, що там є все, що потрібно для роботи з архівами. Він порівняно малого обсягу й не містить нічого зайвого. Зрештою, у ньому відмінно сполучаються простота й функціональність.

Компонент треба ставити в порожню директорію (щоб уникнути жертв), а після установки переконатися (подивившись в tools-> environment options-> вкладка library) у тому, що шлях до цієї самої директорії там прописаний. Інакше потім прийде з'ясовувати, чому компонент є, а нічого не працює, і звідки береться "err_msgs.res". Після установки розглянемо палітру компонентів. Там додалися три закладки (ZTV Tools, ZTV Compress, ZTV Decompress), що містять сумарно 35 компонентів.

ZTV Compress:

– TBlackHole. Призначений для створення blackhole (*.bh) архівів.

– TGZip. Дозволяє створювати gzip архіви..

– TJar, TLha, TCab, TTar. Відповідно, створюють Jar, Lha і Tar архіви.

– TUUEEncode. Створює UUE/XXE кодовані файли для e-mail.

– TZip. Самий головний компонент. Він створює PkZip сумісні ZIP архіви.

Для створення архівів привласнити властивостям ArchiveFile (тип string, це ім'я майбутнього або наявного архіву) і FileSpes (тип TStrings, містить список файлів, що підлягають архівування; також можна використовувати маски типу *.*) відповідні значення, а потім за допомогою методу Compress запустити процес запакування. Він повертає змінну типу integer. Це й буде кількість заархівіруваних файлів. Також можна додати ще й властивість Password. Це буде пароль для архіву. При цьому прописні й малі літери розрізняються.

ZTV Decompress:

– TUnACE, TUnARJ, TUnBH, TUnCab. Розпаковують, відповідно, ACE, ARJ, BH (BlackHole) і MS Cab архіви.

– TUnGZIP. Розпаковує .gz, .z, .tar .jz архіви.

– TUnJar, TUnLha, TunRar, TunTar, TUnZip. Розархівовують Jar (Java Soft формат), Lha, Rar, Tar, ZIP.

Процес розпакування мало чим відрізняється від пакування. Тут необхідно привласнити відповідні значення для ArchiveFile, FileSpec і ExtractDir (визначає директорію, у яку треба розпаковувати) і запустити розпакування за допомогою методу Extract. Цього разу він поверне тобі кількість розпакованих файлів. УВАГА! TUnACE і TUnRAR вимагають для своєї роботи бібліотеки: unace.dll і unrar.dll відповідно.

ZTV Tools**:**

1. TArc2Arc. Конвертує архіви з формату у формат. Для цього нам знадобляться деякі дані.

Властивості:

– ArchiveFile – тут пишеться ім'я старого архіву.

– NewArchive – ім'я нового архіву.

– OutArcType – тип нового архіву; наприклад: tyZip, tyBH, tyLzh, tyCab.

– OutArcMethod – спосіб конвертації.

– IntegrityCheck – чи проводити перевірку нового архіву (true/false).

– DateAttribute – яку дату привласнювати файлам з нового архіву.

Може бути:

– daFileDate – ті ж дати, що й у файлів вихідного архіву.

– daSysDate – системну дату.

– daMaxFileDate – дату самого нового файлу з архіву.

Ця властивість годиться також і всім іншим компонентам.

Методи:

Activate – запускає процес.

2. TMakeSFX. Створює Sfx архів зі звичайного.

Властивості:

– ArchiveFile – ім'я звичайного архіву.

– TargetFile – ім'я майбутнього Sfx'а.

– SfxStubDir – визначає директорію з *.sfx файлами. Вони необхідні для роботи компонента (наприклад, щоб з Zip'а зробити SFX, потрібний ZTV_Zip.SFX), а шлях до них за замовчуванням – c:\windows\system.

Метод CreateSfx запускає весь цей процес.

3. TZipTV. З його допомогою можна з'ясувати практично будь-яку інформацію про архів і файли, що втримуються в ньому.

Властивості:

– ArchiveFile – визначає ім'я архіву.

– FileSpec.Add – маска для запакованих файлів. Тип даних – TStrings.

– FilesInArchive – список заархівіруваних файлів, що відповідає масці з попередньої властивості. Тип теж TStrings.

– Count – кількість запакованих файлів.

– TotalUnpackedSize – вихідний розмір файлів.

– TotalPackedSize – розмір запакованих файлів.

Метод:

GetFileInfo – одержує інформацію про заархівірований файл. Наприклад, таку:

– ArchiveFile – його ім'я.

– Date – дата. реалізується за допомогою функції FormatDateTime. Це робиться, приміром, так: FormatDateTime('mm-dd-yy', ZipTV1.Date). Вона повертає string, що містить дату. Те ж саме й згодом.

– PackedSize – його розмір після пакування.

– UnpackedSize – його вихідний розмір.

– Ratio – compress ratio.

– CRC – його CRC.

4. TZipKey. Компонент допомагає згадати забуті паролі.

– ArchiveFile – ім'я архіву.

– FileSpec – усе та ж маска для файлів.

– CharSets – спосіб пошуку паролів. Наприклад, якщо привласнити Zipkey1.CharSets:= TCharSets(0), то пошук обмежиться паролями, що складаються тільки із цифр. Якщо ж замість нуля буде 8, то пошук буде йти по всім ASCII символах.

– StartPassword – з його починається пошук.

Методи:

– Activate – запускає процес.

– Pause – робить паузу.

Як тільки пароль перебуває, викликається подія OnFound і властивість Password здобуває конкретне значення. Їм і має бути скористатися.

Створення інтерфейсу

Створюється форма й лягає на неї 5 TEdit, 12 TLabel, 7 TButton, 2 TListBox, 2 TProgressBar, 1 TGroupBox.

Опишемо властивості:

– label1 – caption "Відкрити/створити архів:"

– label2 – caption "Пароль для архіву"

– label3 – caption "Шлях для розпакування"

– label4 – caption "Шлях для новонародженого архіву"

– label5 – caption "Усього файлів:"

– label7 – caption "Вихідний розмір:"

– label9 – caption "Розмір після пакування:"

– label11 – caption "Ratio:"

Label'и же під номерами 6, 8, 12 повинні одержати замість caption'a просто нулі, оскільки нормальні значення вони одержать тільки в ран-таймі.

Button'и під номером 1 і 2 одержують caption у вигляді триткрапки. А от інші:

– Button3 – caption "Додати!"

– Button4 – caption "Архівувати"

– Button5 – caption "Розархівувати"

– Button6 – caption "Створити SFX"

– Button7 – caption "Конвертувати"

GroupBox1 одержує caption "Вміст архіву".

Також необхідно визначити на форму компонента: Zip, UnZip, Arc2Arc, MakeSfx і OpenDialog із закладки Dialogs.

Вивід інформації про архів

Якщо користувач вибрав ім'я файлу, ми передаємо його ZipTv, привласнюємо маску *.* і виводимо в ListBox2 файли по цій масці. Тобто всі файли з архіву. Потім виводимо додаткову інформацію про архів: кількість файлів, вихідний розмір даних, пакованний розмір і, нарешті, compress ratio.

Архівування

У першому listbox'е можливо скласти список файлів, що підлягають архівування. У цьому допоможе Edit2, Button2 і Button3 (яка "Додати!", вона

додає вміст Edit2 в ListBox1). Button2 же просто активує OpenDialog і вводить ім'я файлу в Edit2. От як це робиться:

```
OnClick Button2:
ListBox1.Items.Add(Edit2.Text);
OnClick Button3:
ListBox1.Items.Add(Edit2.Text);
```

Втім, маски (типу *.exe, ??e.doc) прийде набивати в Edit2 уже вручну.

Подивимося на Онклік кнопки "Архівувати":

Змінна і буде містити кількість запакованих файлів. Далі ми визначаємо ім'я архіву, співвідносимо вміст ListBox1 із властивістю FileSpec і вставляємо пароль із Edit3. Запускаємо процес і виводимо результат за допомогою ShowMessage.

Розархівування

Тут всі дуже схоже на пакування, крім нової властивості ExtractDir, дані для якого беруться з Edit4.

Створити SFX архів зі звичайного просто. Робиться це наступним чином:

```
MakeSfx1.ArchiveFile:= Edit1.Text;
MakeSfx1.TargetFile:= Edit5.Text;
IF MakeSfx1.CreateSfx then ShowMessage ('Архів створений!');
```

Це і є Онклік від кнопочки "Створити SFX". Шлях до кінцевого файлу буде визначатися вмістом Edit5. Не забудь, що за замовчуванням SFX файли шукаються в системному каталозі.

Конвертація з формату у формат

Тут послідовно визначаються вихідний і кінцевий файли, вибирається спосіб конвертації. Необхідно створити для компонента TZip1 подію OnProgress і вбити туди наступне:

```
ProgressBar1.Position := ProgressByFile;
ProgressBar1.Update;
ProgressBar2.Position := ProgressByArchive;
ProgressBar2.Update;
```

Ці рядки забезпечать користувачеві спостереження за двома важливими процесами: прогресом в архівування поточного файлу й архіву в цілому.

4.2 Захист розробленого програмного забезпечення

Дані які використовуються у даній роботі захищаються алгоритмом ДСТУ 7564:2014 («Купина»). В Україні на основі консервативного підходу із залученням відомих і добре досліджених конструкцій була розроблена ґеш-функція, що базується на новому блоковому шифрі „Калина” (ДСТУ 7624:2014).

Національний стандарт ДСТУ 7564:2014 визначає криптографічну функцію ґешування „Купина”, додатковий режим її застосування для формування коду автентифікації повідомлення (імітовставки), а також значення для перевірки реалізацій.

Для скорочення обсягу тексту національного стандарту була застосована математична нотація, що дозволяє отримати точний і компактний запис.

Водночас, такий підхід може ускладнювати сприйняття сутності алгоритму для фахівців, що не мають фундаментальної криптологічної освіти.

У роботі наводиться розгорнутий альтернативний опис функції ґешування „Купина” із позначеннями, традиційними для галузі комп’ютерних наук.

Термінологія та позначення

Вектор ініціалізації – бітова послідовність фіксованої довжини (512 або 1024 біта), що використовується як початкове значення при обчисленні ґеш-значення.

Внутрішній стан – бітова послідовність фіксованої довжини (512 або 1024 біта), що є проміжним значенням на кожній ітерації перетворення функції ґешування, а також вхідним та вихідним значеннями перетворень P і Q; для цих перетворень внутрішній стан подається як матриця розміром $8\times c$ байт.

Ґеш-значення (ґеш-вектор) – бітова послідовність фіксованої довжини ($n = 8\cdot s$, $s \in \{1,2,...,64\}$), що є результатом роботи функції ґешування.

Доповнення – вставка додаткових біт у кінець повідомлення для отримання кратності довжини бітової послідовності довжині внутрішнього стану функції ґешування.

Повідомлення – бітова послідовність довжини від 0 біт (порожній рядок) до 2^{96}-1 біт.

Функція стиснення – ітеративне перетворення, що відображає *l*-бітний блок повідомлення та *l*-бітне значення, отримане функцією стиснення на попередньому кроці, у нове *l*-бітне значення.

Далі використовуються наступні позначення:

– ⊕ – додавання за модулем 2 (XOR);

– 0x – префікс числа, що записане у шістнадцятковій системі числення;

– *a*mod*b* – ціле невід'ємне число, що дорівнює залишку від ділення цілого числа *a* на натуральне число *b*;

– B_i – *i* -й байт вхідної послідовності;

– C^i – константа перетворення XORRoundKey або Add64RoundKey для *i* -го циклу;

– *c* – кількість стовпців внутрішнього стану в матричному поданні;

– φ – функція стиснення;

– *H* – визначена у стандарті функція ґешування;

– *H*(*M*) – результат обчислення функції ґешування для повідомлення *M* (ґешзначення);

– IV – вектор ініціалізації;

– *l* – розмір внутрішнього стану функції ґешування (у бітах), $l \in \{512,1024\}$;

– *M* – повідомлення;

– m_i – *i* -й блок повідомлення *M*;

– *n* – довжина обчисленого ґеш-значення;

– *N* – довжина повідомлення *M* без доповнення;

– *P*, *Q* – складові перетворення функції стиснення;

– P_{512} – перетворення *P* для 512-бітного внутрішнього стану;

– P_{1024} – перетворення *P* для 1024-бітного внутрішнього стану;

– Q_{512} – перетворення *Q* для 512-бітного внутрішнього стану;

– Q_{1024} – перетворення Q для 1024-бітного внутрішнього стану;

– r – кількість ітерацій у перетвореннях P і Q ($r = t$ в ДСТУ 7564:2014);

– S – внутрішній стан ґеш-функції;

– t – кількість блоків m, з яких складається повідомлення M, включаючи доповнення;

– v_i – i -й біт вхідної послідовності;

– Ω – завершальне перетворення;

– Купина-n – режим використання функції ґешування з усіченням обчисленого ґешзначення до розміру n біт.

Загальні положення

Під функцією ґешування H розуміється залежне від вектора ініціалізації IV відображення послідовності біт M у ґеш-значення $H(M)$ фіксованої довжини n.

ДСТУ 7564:2014 визначає функцію ґешування, яка виконує перетворення «Купина-256» або «Купина-512», що забезпечують обчислення ґеш-значення з довжинами 256 або 512 біт відповідно.

Ґеш-значення довжиною 256 бітів додатково може бути усічено до бітової послідовності довжиною від 8 до 248 біт з кроком у 8 біт, 512 бітів може бути усічене до бітової послідовності довжиною від 264 до 504 біт з кроком у 8 біт.

Режим роботи для формування ґеш-значення довжиною n біт позначається як «Купина-n».

Основними режимами роботи функції ґешування, що рекомендуються до застосування, є «Купина-256», «Купина-384» і «Купина-512».

Структура перетворення

Функція ґешування, визначена в ДСТУ 7564:2014, формує ґеш-значення для повідомлення, що складається з бітової послідовності довжини від 0 біт (порожній рядок) до 2^{96}-1 біт.

При формуванні ґеш-значення повідомлення доповнюється, далі поділяється на l-бітні блоки $m_0,\ldots, m_t$, після чого виконується обробка кожного блоку шляхом ітеративного виконання функції стиснення φ.

При цьому формуються значення $h_i = \varphi(h_{i-1}, m_i)$ де $i = 1,...,t$, а початкове значення $h_0 = \text{IV}$.

Після обробки останнього блоку повідомлення результуюче ґеш-значення обчислюється як $H(M) = \Omega(h_t)$, де Ω – завершальне перетворення, що повертає n -бітне значення, кратне 8 ($0 < n \leq l/2$).

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ КІБЕРБЕЗПЕКИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображене головне вікно програми. З нього ми бачимо, що у програмі є 5 випадаючих меню:

– Файл.

– Архівувати.

– Розархівувати.

– Параметри.

– Довідка.

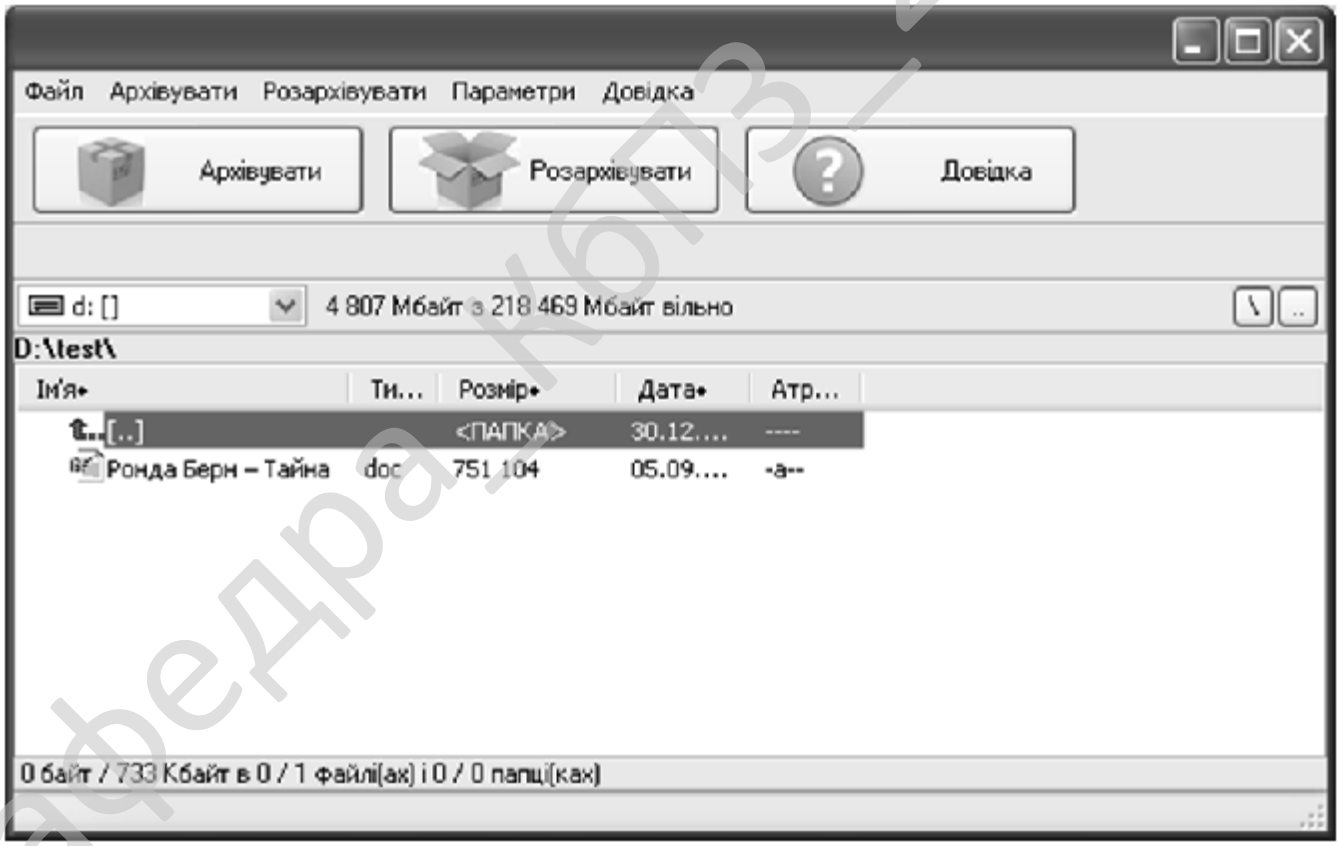

Рисунок 5.1 – Основне вікно програми

Також на головній панелі існують три кнопки, при натисненні на які можливе проведення наступних дій:

– Архівувати файли.

– Розархівувати файли.

– Отримати довідку.

При виборі процедури архівування, на екран виводиться процес, вікно якого зображено на рисунку 5.2.

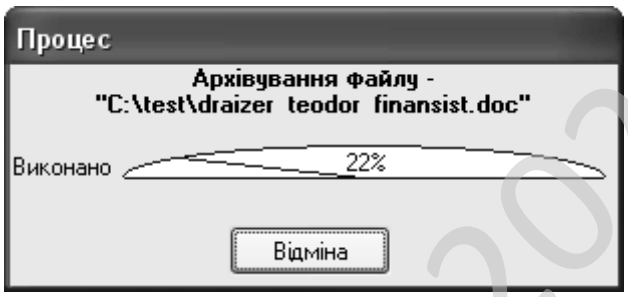

Рисунок 5.2 – Процес архівування

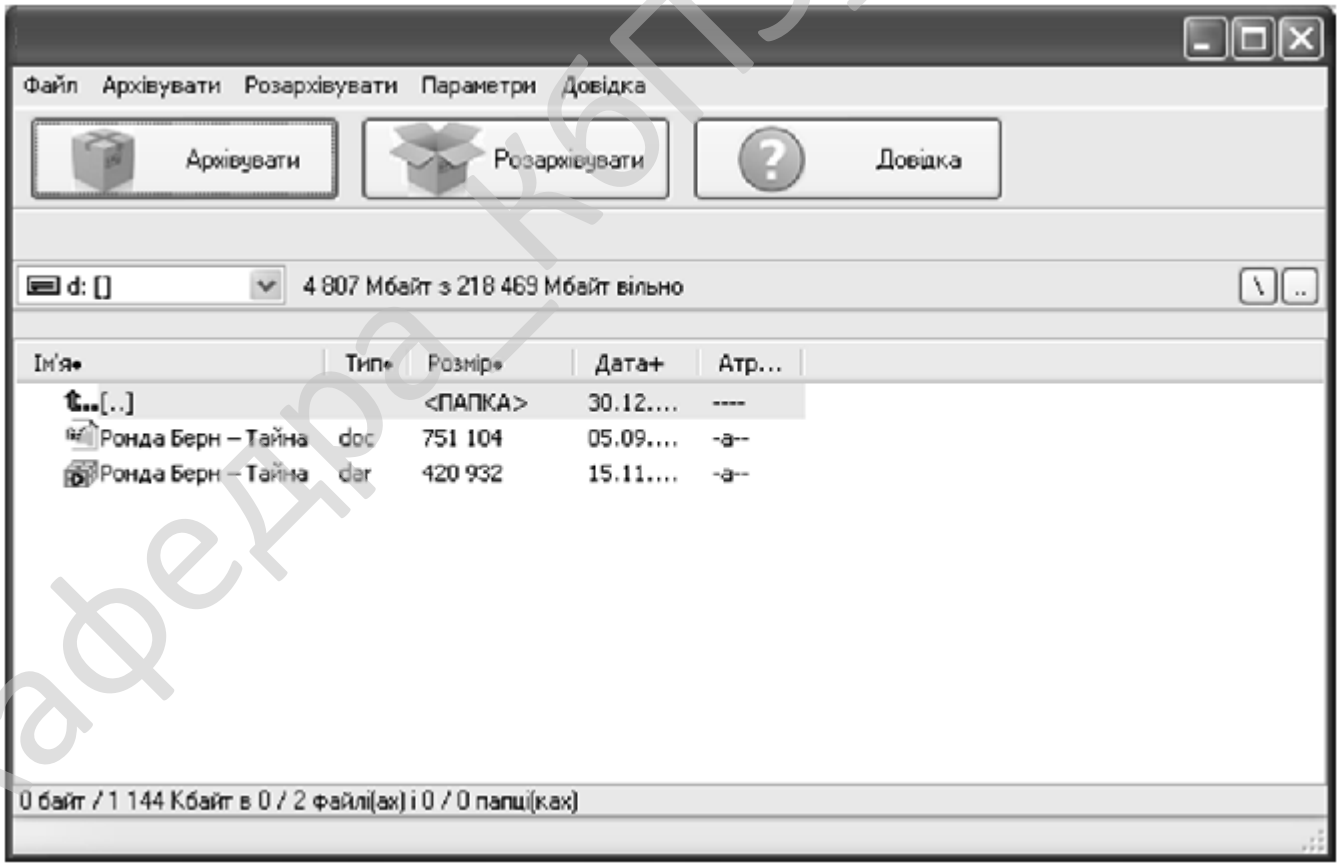

Рисунок 5.3 – Результат архівування

Результат процесу архівування зображено на рисунку 5.3.

При натисканні на кнопку розархівування, запускається процес зображений на рисунку 5.4. На рисунку 5.5 наведено меню Довідка.

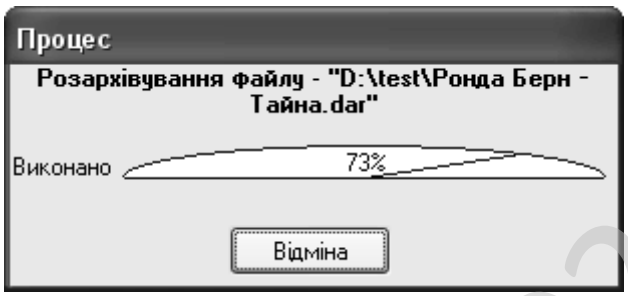

Рисунок 5.4 – Процес розархівування

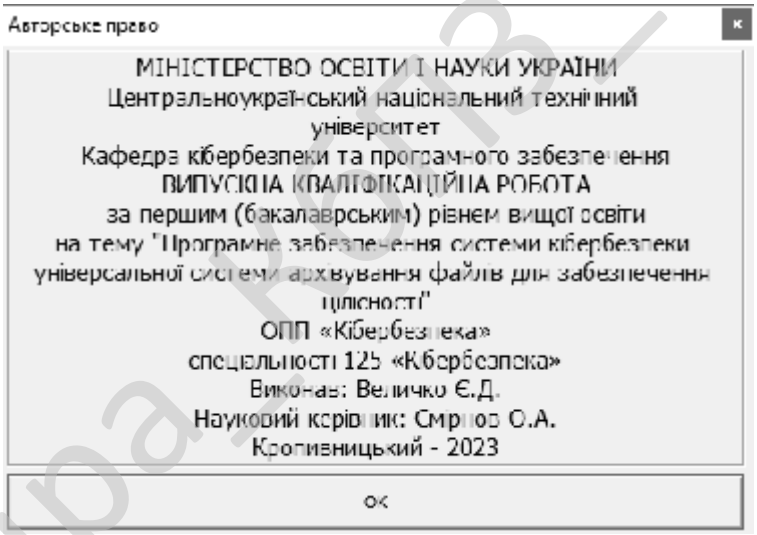

Рисунок 5.5 – Довідка

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи кібербезпеки універсальної системи архівування файлів для забезпечення цілісності.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем універсальної системи архівування файлів для забезпечення цілісності.

– Досліджена система універсальної системи архівування файлів для забезпечення цілісності.

– На основі отриманих результатів досліджень створена програмна реалізація системи кібербезпеки універсальної системи архівування файлів для забезпечення цілісності.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання універсальної системи архівування файлів для забезпечення цілісності.

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 RAD Studio Delphi 10.4. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи кібербезпеки універсальної системи архівування файлів для

забезпечення цілісності. 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 кібербезпеки й включає програми, що реалізують її функції.

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм ДСТУ 7564:2014.

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. 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, pp. 1-12. **(Scopus).**

2. Smirnov O., Smirnova T., Anas M. Al-Oraiqat, Drieiev O., Polishchuk L., Sheroz Khan, Yassin M. Y. Hasan, Aladdein M. Amro, Hazim S. AlRawashdeh «Method for Determining Treated Metal Surface Quality Using Computer Vision Technology». *Sensors (Basel, Switzerland)* Volume 22, Issue 16, 6223, 2022. (**Scopus**).

3. 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. **(Scopus).**

4. 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. **(Scopus).**

5. Smirnov O., Kuznetsov A., Kryvinska N., Kiian A., Kuznetsova K. «Full Non-Binary Constant-Weight Codes». *SN Computer Science,* Vol 2, 337, 2021. https://doi.org/10.1007/s42979-021-00739-w **(Scopus).**

6. Smirnov O., Kovalenko O., Kovalenko A., Kavun S. «Quantitative Risk Assessment Method Development in the Context of the SDLC-model». *2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T),* 2021, pp. 203-208, doi: 10.1109/PICST54195.2021.9772143 **(Scopus).**

7. Smirnov O., Neskorodieva T., Fedorov E., Rymar P. «Neural Network Modeling Method of Transformations Data of Audit Production with Returnable Waste». *CEUR Workshop Proceedings* Volume 3101, 2021, Pages 192-207. **(Scopus).**

8. 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.* **(Scopus)**.

9. 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. **(Scopus)**.

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

11. Smirnov O.A., 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. **(Scopus)**.

12. 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. **(Scopus)**.

13. 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. **(Scopus)**.

14. 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. **(Scopus)**.

15. 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. **(Scopus)**.

16. 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. **(Scopus)**.

17. Smirnov, O., Drieieva, H., Drieiev, O., Polishchuk, Y., Brzhanov, R., Aleksander, M. «Method of fractal traffic generation by a model of generator on the graph». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 366-379. **(Scopus)**.

18. 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. **(Scopus)**.

19. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», *CEUR Workshop Proceedings* Volume 2608, 2020, Pages 633-645. **(Scopus)**.

20. 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., **(Scopus)**.

21. 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. (**Scopus)**.

22. 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. **(Scopus)**.

23. 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. **(Scopus)**.

24. 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. **(Scopus)**.

25. 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. **(Scopus)**.

26. Smirnov, O., Kuznetsov, A., Prokopovych-Tkachenko, D. «Hiding Data in Images Using a Pseudo-Random Sequence». *ISCI'2020: Information Security in Critical Infrastructures.* ***Collective monograph***. Edited by Ivan D. Gorbenko, Victor A. Krasnobayev and Alexandr A. Kuznetsov. ASC Academic Publishing, USA, 2020. pp. 46-59. – ISBN: 978-1-7362833-0-1 (Hardback), ISBN: 978-1-7362833-1-8 (Ebook).

27. Smirnov, O., Kuznetsov, A., Shekhanin, K., Chepurko, I. Detecting Hidden Information in FAT. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures**. Collective monograph**. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 412-429. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

28. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures. **Collective monograph**. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

29. О.А. Смірнов, П.С. Усік, «Дослідження перспектив використання технологічних рішень в мережах 5G» у *Кібербезпека та інформаційні технології: монографія.* – Х. : ТОВ «ДІСА ПЛЮС», 2020.С. 122-135.

30. Смірнов О.А., Дрєєва Г.М., «Метод генерування фрактального трафіку за допомогою моделі генератора на графі» у Інформаційна безпека та інформаційні технології: монографія / за заг. ред. В. С. Пономаренка. – Х. : Вид. Рожко С.Г. 2019. С. 123-139.

31. Смирнов А.А., Коваленко А.В. Комплекс математических моделей технологии тестирования WEB-приложений. Інформаційні технології: сучасний стан та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: ТОВ «ДІСА ПЛЮС», 2018. – 461 с.

32. Смирнов А.А., Коваленко А.В. Разработка метода управления рисками разработки программного обеспечения. Інформаційні технології: проблеми та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: Видавець Рожко С.Г., 2017. – 447 с.

33. 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. 2022.

34. 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.

35. 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.

36. Смірнов О.А., Смірнова Т.В., Буравченко К.О., Кравченко С.С., Горбов В.О., «Хмарна система підтримки прийняття рішень технологічного

процесу відновлення поверхонь конструкцій і деталей машин». *Сучасні інформаційні системи*. 2021. Т. 5, № 4. С. 79-95

37. Смирнов А., Кузнецов А., Кузнецова Т. «Шумоподобные дискретные сигналы для асинхронных систем кодового разделения радиоканалов». *Радиотехника*, № 2(205), 175–183. 2021.

38. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New Technique for Hiding Data in Cover Images Using Adaptively Generated Pseudorandom Sequences». *CEUR Workshop Proceedings* Volume 2732, 2020, Pages 214-227.

39. Смірнов, О.А., Полігенько О.О., Одарченко Р.С., Терещенко Л.Ю.Усік П.С., «Інформаційна технологія та програмне забезпечення для підвищення ефективності планування підсистеми базових станцій стільникового зв'язку». *Проблеми телекомунікацій*. № 1(26). С. 83-96. 2020.

40. Смирнов А.А, Кузнецов А.А., Киян А.С., Кузнецова Е.А. «Сокрытие данных на основе адресации шумоподобных сигналов». *Всеукраїнський міжвідомчий науково-технічний збірник "Радіотехніка"* – Харьків: ХНУРЕ. – 2020. – Вип. 203. – С. 38-49.

41. Смирнов А.А., Дудан А.В., Смирнова Т.В. «Формализация структуры технологического процесса электродугового напыления». *Сборник научных трудов «Актуальные вопросы машиноведения».* Объединенный институт машиностроения Национальной Академии Наук Беларуси. №9. С. 308-312, 2020.

42. Смірнов О.А., Усік П.С., Миронець І.В., Буравченко К.О., Якименко Н.М. «Метод підвищення ефективності розподіленої обробки даних у комп'ютерних системах операторів стільникового зв'язку» *Вісник Черкаського державного технологічного університету. Технічні науки.* №4. С. 103-110. 2020.

43. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», *Кібербезпека: освіта, наука, техніка*. № 3(7). С. 43-62. 2020.

44. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». *Центральноукраїнський науковий вісник. Технічні науки.* № 2(33). с. 161-172, 2019.

45. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., «Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей». Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

46. Смірнов О.А., Смірнова Т.В., Солових Є.К., Дрєєв О.М., «Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей». Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

47. Смірнов О.А., Смірнова Т.В.,. Дрєєв О.М, «Експертна система оптимізації процесу відновлення та зміцнення поверхонь деталей типу «вал» електродуговим напиленням», Системи управління, навігації та зв'язку, № 2 (54). с. 149-154, 2019.

48. Смірнов О.А., Смірнов С.А., Поліщук Л.І., Смірнова Т.В., Конопліцька-Слободенюк О.К. Метод формування антивірусного захисту даних з використанням безпечної маршрутизації метаданих. Кібербезпека: освіта, наука, техніка. – Том 3 № 3. – Київ: КУ ім. Бориса Грінченка. – 2019. – С. 63-87.

49. Смирнов А.А., Лысенко И.А., Информационная технология проектирования тестовых наборов на основе требований к программному обеспечению, Системи управління, навігації та зв'язку. – Випуск 4 (44). – Полтава: ПолтНТУ. – 2017. – С. 112-115.

50. Смірнов О.А., Мелешко Є.В., Хох В.Д., Дослідження методів аудиту систем управління інформаційною безпекою, Системи управління, навігації та зв'язку. – Випуск 1 (41). – Полтава: ПолтНТУ. – 2017. – С. 38-42.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки універсальної системи архівування файлів для забезпечення цілісності.

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. № 12-02 від 5.01.2023 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи кібербезпеки універсальної системи архівування файлів для забезпечення цілісності.

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 кібербезпеки з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи кібербезпеки універсальної системи архівування файлів для забезпечення цілісності;

– 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 архітектури IBM PC, 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 RAD Studio Delphi 10.4.

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 – 88 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 23.05.2023 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 8.06.2023 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

_____ Смірнов О.А.

Програмне забезпечення системи кібербезпеки універсальної системи архівування файлів для забезпечення цілістності

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 54

Літера: РП

Кропивницький – 2023 року

Файл Pr_arh.dpr – головний файл проекту

```
program Pr_arh;

uses
  Forms,
  Main in 'Main.pas' {fmPR_ARH}, //головне вікно
  About in 'About.pas' {fmAbout},//файл інформації про розробника
  frFilePanelU in 'frFilePanelU.pas' {frFilePanel: TFrame},//панелі
  FilesExU in 'FilesExU.pas',
  FilesU in 'FilesU.pas',
  fmAnyMessageU in 'fmAnyMessageU.pas' {fmAnyMessage},//форма повідомлень
  fmErrorDriveU in 'fmErrorDriveU.pas' {fmErrorDrive},//форма помилок
  fmNameQueryU in 'fmNameQueryU.pas' {fmNameQuery},//форма запитів
  StrConsts in 'StrConsts.pas',// константи
  DeCompressorU in 'DeCompressorU.pas',//алгоритми архівування
  DFileStreamU in 'DFileStreamU.pas', //алгоритми розархівування
  DictionaryU in 'DictionaryU.pas',//словник
  ArhU.pas in 'Arh.pas',//форма архівування
  fmExtractDirU in 'fmExtractDirU.pas' {fmExtractDir},//форма добування з архіву
  fmProcessU in 'fmProcessU.pas' {fmProcess};//форма процесу роботи

{$R *.RES}
//основна программа
begin
  Application.Initialize;
  Application.Title := 'Архіватор';
  Application.CreateForm(TfmPR_ARH, fmPR_ARH);//створення вікна архівування
  Application.CreateForm(TfmAbout, fmAbout); //створення вікна інформації про
розробника
  Application.CreateForm(TfmAnyMessage, fmAnyMessage); //створення вікна
виведення повідомлень
  Application.CreateForm(TfmErrorDrive, fmErrorDrive); //створення вікна
виведення помилок
  Application.CreateForm(TfmNameQuery, fmNameQuery); //створення вікна запитів
  Application.CreateForm(TfmExtractDir, fmExtractDir); //створення вікна папки
куди розахівовувати
  Application.CreateForm(TfmProcess, fmProcess); //створення вікна процесу роботи
  Application.Run; //створення вікна початку роботи
end.
```

Файл Arh.pas - реалізація роботи архіватора

```
unit Arh;

{$H+}

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Grids, XPMan, Math;

type
  TForm1 = class(TForm)
    Button1: TButton;
    OpenDialog1: TOpenDialog;
    SG1: TStringGrid;
    Button2: TButton;
    StringGrid1: TStringGrid;
    Button3: TButton;
    SaveDialog1: TSaveDialog;
    procedure Button1Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormCreate(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

TStr = string[20];
TKey = string[1];
TStack = array[0..65000] of TKey;

TPElSp = ^TElSp;
TElSp = record
  key: byte;
  cnt: cardinal;
  next: TPElSp;
  end;

TPArNode = ^TArNode;
TArNode = record
  value: real;
  kod: TKey;
  left: TPArNode;
  right: TPArNode;
  parent: TPArNode;
  symb: byte;
  end;

TTree = record
  key: byte;
  cnt: cardinal;
  huff: TStr;
  n0: extended;
  deep: boolean;
  st_tree: TPArNode;
  end;

TArTree = array of TTree;

TFileRec = record
  key: byte;
```

```
  kod: byte;
  size_kod: byte;
  end;

TGetFile = record
  key: byte;
  kod: string[16];
  size_kod: byte;
  end;

TArFRec = array of TGetFile;

var
  stack: TStack;
  PBS: integer;
  Form1: TForm1;
  head_lsp: TPElSp;
  f, fo: file;
  size, tmpi, cnti: byte;
  a, atmp: TArTree;
  n: int64;
  head_tree: TPArNode;
  okk: boolean;
implementation

{$R *.dfm}

procedure Push(key: TKey);
begin
 stack[PBS]:= key;
 inc(PBS);
end;

procedure Pop(var key: TKey);
begin
 dec(PBS);
 key:= stack[PBS];
end;

procedure AddEl(var start: TPElSp; PNew: TPElSp);
var
 WP: TPElSp;
begin
 PNew^.next:= nil;
 if start = nil then start:= PNew
 else
  begin
  WP:= start;
  while WP^.next <> nil do
   WP:= WP^.next;
  WP^.next:= PNew;
end;
end;

function FindEl(start: TPElSp; key: byte; var FindPoint: TPElSp): boolean;
begin
 if start = nil then
 begin
  result:= false;
  exit;
 end;
 result:= false;
 FindPoint:= start;
 while (FindPoint <> nil) and (FindPoint^.key <> key) do
  begin
  findpoint:= findpoint^.next;
  end;
  if (findpoint <> nil) then result:= true;
end;
```

```
function FindElMax(start: TPElSp; var FindPoint: TPElSp): boolean;
var
 PrevPoint: TPElSp;
begin
 if start = nil then
 begin
  result:= false;
  exit;
 end;
 FindPoint:= start;
 PrevPoint:= start^.next;
 while (PrevPoint <> nil) do
  begin
  if PrevPoint^.cnt > FindPoint^.cnt then
   FindPoint:= PrevPoint;
  Prevpoint:= PrevPoint^.next;
  end;
  result:= true;
end;

procedure DelEl(var start: TPElSp; key: byte);
var
PrevPoint, WPoint: TPElSp;
begin
if start = nil then exit;
 prevpoint:= nil;
 wpoint:= start;
 while (wpoint <> nil) and (wpoint^.key <> key) do
 begin
  prevpoint:= wpoint;
  wpoint:= wpoint^.next;
 end;
 if (wpoint = nil) or (wpoint^.key > ord(key)) then
 exit;
 if prevpoint = nil then
  start:= start^.next
 else
  prevpoint^.next:= wpoint^.next;
  Dispose(Wpoint);
end;

procedure DelSp(var head: TPElSp);
var
 w: TPElSp;
begin
  if head = nil then exit;
  while (head <> nil) do
  begin
  w:= head;
  head:= head^.next;
  Dispose(w);
  end;
end;

function FindMin(a: TArTree; size: integer): integer;
var
 z, i: integer; min: extended; ok: boolean;
begin
 ok:= false;
 for z:= size-1 downto 0 do
 begin
  if a[z].deep <> true then
  begin
  min:= a[z].n0;
  result:= z;
  ok:= true;
  i:= z-1;
  end;
```

```
  if ok = true then
   break;
 end;
  for z:= i downto 0 do
   if (a[z].n0 < min) and (a[z].deep <> true) then
   begin
    min:= a[z].n0;
    result:= z;
   end;
end;

function StrBToInt(bin: string): byte;
var
i: integer; v: byte;
begin
  v:= 0;
  result:= 0;
  for i:=length(bin) downto 1 do
  begin
    result:= result + StrToInt(bin[i]) * StrToInt(FloatToStr(exp(ln(2)*v)));
    inc(v);
  end;
end;

function IntToStr(in_int: byte): string;
var
 k: TKey;
 t1, t2: byte;
begin
  PBS:= 0;
  t1:= in_int;
  t2:= in_int;
  result:= '';
  while (t2 <> 1) and (t2 <> 0) do
  begin
   t1:= t2 mod 2;
   k:= IntToStr(t1);
   push(k);
   t2:= t2 div 2;
  end;
   k:= IntToStr(t2);
   push(k);
  while (PBS <> 0) do
  begin
   pop(k);
   result:= result + k;
  end;
end;

function FindElAr(ar: TArTree; size: byte; key: byte): byte;
var
 i: integer;
begin
 for i:= 0 to size-1 do
  if ar[i].key = key then
  begin
   result:= i;
   exit;
  end;
end;

function FindElAr(ar: TArFRec; size: byte; key: byte): byte;
var
 i: integer;
begin
 for i:= 0 to size-1 do
  if ar[i].key = key then
  begin
   result:= i;
```

```
   exit;
  end;
end;


procedure reverse(var pesn: TStr);
var
 i: integer; tmp: char;
begin
for i:=1 to (length(pesn) div 2) do
begin
    tmp:=pesn[i];
    pesn[i]:=pesn[length(pesn)-i+1];
    pesn[length(pesn)-i+1]:= tmp;
end;
end;

procedure CreateKod(head: TPArNode; var ar: TArTree; size: byte);
var
 k: TKey;
begin
 if (head = nil) then exit;
   push(head^.kod);
   CreateKod(head^.left, ar, size);
   tmpi:= FindElAr(ar, size, head^.symb);
   cnti:=0;
   while (PBS<>0) do
   begin
    inc(cnti);
    pop(k);
    ar[tmpi].huff:= ar[tmpi].huff + k;
   end;
   reverse(ar[tmpi].huff);
   PBS:= PBS + cnti;
   CreateKod(head^.right, ar, size);
   dec(PBS);
end;

procedure GetNode(a: TArTree; size: integer);
var
 tmp: TPArNode;
begin
 tmp:= a[ size-1].st_tree;
 while (tmp^.parent <> nil) do
  tmp:= tmp^.parent;
 head_tree:= tmp;
end;

function EndHaffman(a: TArTree; size: integer): extended;
var
 tmp: TPArNode;
begin
 tmp:= a[ size-1].st_tree;
 while (tmp^.parent <> nil) do
  tmp:= tmp^.parent;
 result:= tmp^.value;
end;

procedure OptimizeKod(var ar: TArTree; size: byte);
var
 i, j: integer; ok: boolean; str: string; news, tmp: string[1];
begin
 for i:= 0 to size-1 do
 begin
  ok:= true;
  SetLength(str, length(ar[i].huff));
  str:= ar[i].huff;
  tmp:= str[1];
  if tmp = '0' then
```

```
  begin
   for j:=1 to length(str) do
    if (str[j] <> '0') then
     ok:= false;
   end else
     ok:= false;
   if ok then
   begin
   SetLength(news, length(ar[i].huff));
   for j:=1 to length(ar[i].huff) do
    news[j]:= '1';
    ar[i].huff:= news;
   end;
 end;
end;

function GetText(ar: TArTree; size: byte): widestring;
var
 i: byte;
begin
result:= '';
 for i:= 0 to size-1 do
  result:= result + a[i].huff;
end;

procedure Countall(a: TArTree; size: byte);
var
 i: integer;
 H, Hmax, LiPi, Kss, Koe: real;
begin
  Hmax:= log2(size);
  H:= 0;
  LiPi:= 0;
  for i:= 0 to size - 1 do
  begin
   H:= H + a[i].n0 * log2(a[i].n0);
   LiPi:= LiPi + length(a[i].huff) * a[i].n0;
  end;
  H:= (-1)*H;
  liPi:= log2(LiPi);
  Kss:= Hmax / LiPi;
  Koe:= H / LiPi;
  Form1.StringGrid1.Cells[1,0]:= FloatToStr(Hmax);
  Form1.StringGrid1.Cells[1,1]:= FloatToStr(H);
  Form1.StringGrid1.Cells[1,2]:= FloatToStr(LiPi);
  Form1.StringGrid1.Cells[1,3]:= FloatToStr(Kss);
  Form1.StringGrid1.Cells[1,4]:= FloatToStr(Koe);
end;

procedure TForm1.Button1Click(Sender: TObject);
var
 s, tmps: string;
 tmp: char;
 i, i1, i2, j: cardinal;
 tpos: integer;
 El, tel: TPElSp;
 ElTree, tmp1, tmp2: TPArNode;
 lvl, outkod, inkod: word;
 f1, H, LiPi: extended;
 rec: TFileRec;
 res: widestring;
 str: string;
 chk: byte;
begin
  Opendialog1.FilterIndex:= 1;
  head_lsp:= nil;
  head_tree:= nil;
  size:= 0;
  n:= 0;
```

```
PBS:= 0;
SetLength(a, 0);
OpenDialog1.Execute;
if OpenDialog1.FileName = '' then
begin
  ShowMessage('Не обрано файл!');
  exit;
end;
AssignFile(f, OpenDialog1.FileName);
Reset(f, 1);
while not eof(f) do
begin
 BlockRead(f, tmp, 1);
 if FindEl(head_lsp, ord(tmp), tel) = false then
 begin
 New(El);
 El^.key:= ord(tmp);
 El^.cnt:= 1;
 AddEl(head_lsp, El);
 inc(size);
 inc(n);
 end else
 begin
 inc(tel^.cnt);
 inc(n);
 end;
end;
SetLength(a, size);
i:= 0;
while (head_lsp <> nil) do
begin
  FindElMax(head_lsp, tel);
  a[i].key:= tel^.key;
  a[i].cnt:= tel^.cnt;
  a[i].n0:= a[i].cnt / n;
  a[i].deep:= false;
  a[i].huff:='';
  new(ElTree);
  ElTree^.value:= 0;
  ElTree^.left:= nil;
  ElTree^.right:= nil;
  ElTree^.parent:= nil;
  ElTree^.kod:= '';
  ElTree^.symb:= a[i].key;
  a[i].st_tree:= ElTree;
  inc(i);
  DelEl(head_lsp, tel^.key);
end;
while (EndHaffman(a, size) <> 1) do
 begin
  i1:= FindMin(a, size);
  a[i1].deep:= true;
  i2:= FindMin(a, size);
  a[i2].n0:= a[i2].n0 + a[i1].n0;
  new(ElTree);
  ElTree^.value:= a[i2].n0;
  ElTree^.parent:= nil;
  ElTree^.kod:='';
  tmp1:= a[i1].st_tree;
  while (tmp1^.parent <> nil) do
   tmp1:= tmp1^.parent;
  tmp2:= a[i2].st_tree;
  while (tmp2^.parent <> nil) do
   tmp2:= tmp2^.parent;
  ElTree^.left:= tmp1;
  ElTree^.right:= tmp2;
  tmp1^.parent:= ElTree;
  tmp1^.kod:= '0';
  tmp2^.kod:= '1';
```

```
    tmp2^.parent:= ElTree;
  end;
  GetNode(a, size);
  CreateKod(head_tree, a, size);
  seek(f, 0);
  assignfile(fo, 'test.asd');
  rewrite(fo, 1);
  BlockWrite(fo, size, 1);
  BlockWrite(fo, size, 1);
  for i:=0 to size-1 do
  begin
   rec.key:= a[i].key;
   rec.kod:= StrBToInt(a[i].huff);
   rec.size_kod:= length(a[i].huff);
   BlockWrite(fo, rec, sizeof(TFileRec));
  end;
  j:= 0;
   res:= '';
   while (not eof(f)) do
   begin
     tpos:= filepos(f);
     BlockRead(f, inkod, 1);
     i:= FindElAr(a, size, inkod);
     res:= res + a[i].huff;
     inc(j);
   end;
   while length(res)<>0 do
   begin
    i:= 0;
    str:= '';
    while (i<>8) and (i<>length(res)) do
    begin
     inc(i);
     str:= str + res[i];
    end;
    delete(res, 1, i);
    outkod:= StrBToInt(str);
    BlockWrite(fo, outkod, 1);
   end;
    seek(fo, 1);
    chk:= length(str);
    BlockWrite(fo, chk, 1);
  Countall(a, size);
  for i:=0 to size-1 do
  begin
    SG1.Cells[0,i+1]:= IntToStr(a[i].key);
    SG1.Cells[1,i+1]:= chr(a[i].key);
    SG1.Cells[2,i+1]:= IntToStr(a[i].cnt);
    SG1.Cells[3,i+1]:= IntToStr(length(a[i].huff));
    SG1.Cells[4,i+1]:= a[i].huff;
    SG1.RowCount:= SG1.RowCount + 1;
  end;
    SG1.RowCount:= SG1.RowCount - 1;
    closefile(f);
    closefile(fo);
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 DelSp(head_lsp);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  PBS:= 0;
  SG1.Cells[0,0]:= 'Код';
  SG1.Cells[1,0]:= 'Символ';
  SG1.Cells[2,0]:= 'Частота';
  SG1.Cells[3,0]:= 'Довжина';
```

```
  SG1.Cells[4,0]:= 'Код Хаффмана';
  StringGrid1.Cells[0,0]:= 'H';
  StringGrid1.Cells[0,1]:= 'Hmax';
  StringGrid1.Cells[0,2]:= 'Lcp';
  StringGrid1.Cells[0,3]:= 'Кс.с.';
  StringGrid1.Cells[0,4]:= 'К.е.';
end;

function GetFromKod(a: TArFRec; size: byte; key: string; var pos: byte):
boolean;
var
 i: byte;
begin
 result:= false;
 pos:= 0;
 for i:=0 to size-1 do
  if a[i].kod = key then
  begin
   result:= true;
   pos:= i;
   exit;
  end;
end;

function CorrectKod(str: string; itSize: byte): string;
var
 i, j, k: integer; tmp: string[16];
begin
  if length(str) = itSize then
  begin
   result:= str;
   exit;
  end;
  tmp:= '0000000000000000';
  i:= itSize - length(str);
  delete(tmp, itSize+1, 16);
  k:= 1;
  for j:=i+1  to itSize do
  begin
    tmp[j]:= str[k];
    inc(k);
  end;
  result:= tmp;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
fin, fout: file; arr: TArFRec; i, j, ind:integer;
 size1, pos, tmp, chk: byte;
rec: TFileRec; data: byte; res, str: widestring;
tmps: string;
begin
  Opendialog1.FilterIndex:= 2;
  size1:= 0;
  OpenDialog1.Execute;
  if OpenDialog1.FileName = '' then
  begin
    ShowMessage('Не обраний вихідний файл!');
    exit;
  end;
  AssignFile(fin, OpenDialog1.FileName);
  Reset(fin, 1);
  SaveDialog1.Execute;
  if SaveDialog1.FileName = '' then
  begin
    ShowMessage('Не обраний файл для архівації!');
    exit;
  end;
  AssignFile(fout, SaveDialog1.FileName);
```

```
  rewrite(fout, 1);
  BlockRead(fin, size1, 1);
  BlockRead(fin, chk, 1);
  SetLength(arr, size1);
  for i:= 0 to size 1-1 do
  begin
   BlockRead(fin, rec, sizeof(TFileRec));
   arr[i].key:= rec.key;
   arr[i].kod:= CorrectKod(IntToStr(rec.kod), rec.size_kod);
   arr[i].size_kod:= rec.size_kod;
  end;
   res:= '';
   while (not eof(fin)) do
   begin
     if ((filepos(fin) + 1) <> filesize(fin)) then
     begin
      BlockRead(fin, data, 1);
      tmps:= CorrectKod(IntToStr(data), 8);
      res:= res + tmps;
      inc(j);
     end else
     begin
      BlockRead(fin, data, 1);
      tmps:= CorrectKod(IntToStr(data), chk);
      res:= res + tmps;
     end;
   end;
   while length(res)<>0 do
   begin
    i:= 0;
    str:= '';
    while (GetFromKod(arr, size1, str, pos)=false) and (i<>length(res)) do
    begin
     inc(i);
     str:= str + res[i];
    end;
    delete(res, 1, i);
    BlockWrite(fout, arr[pos], 1);
  end;
end;

end.
```

Файл Main.pas - головне вікно програми

```
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.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 собе', '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 - кодування

```
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 довгий}
  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  //то нічого не робимо
    Out:=-1024;
    InpWord:=New;  //Змінимо вхідне слово
  End
  Else Begin //Якщо немає в словнику, то...
    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, то 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 - декодування

```
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 довгий}
  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  //то нічого не робимо
    Out:=-1024;
    InpWord:=New;  //Змінимо вхідне слово
  End
  Else Begin //Якщо немає в словнику, то...
    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, то 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 користувача

```
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-}
  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-}
    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-}
  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-}
    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-}
{$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-}
  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('БАКАЛАВРСЬКИЙ ПРОЕКТ');
Memo1.Lines.Add('');
Memo1.Lines.Add('на  тему:');
Memo1.Lines.Add('');
Memo1.Lines.Add(' Програмне забезпечення системи кібербезпеки універсальної
системи архівування файлів для забезпечення цілістності');
Memo1.Lines.Add('');
Memo1.Lines.Add('Керівник:  Смірнов О.А.');
Memo1.Lines.Add('');
Memo1.Lines.Add('Розробив:  студент   Величко Євгеній Дмитрович');
Memo1.Lines.Add('                     гр. КБ-19');
Memo1.Lines.Add('');
Memo1.Lines.Add('м. Кропивницький 2023');
Memo1.Lines.Add('');
end;

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