

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ ЗАХИСТУ ІНФОРМАЦІЇ

ЛАБОРАТОРНА РОБОТА №1

з дисципліни

«Алгоритми кодування двійкових даних»

на тему: «Реалізація алгоритмів побайтного перетворення інформації
(Base64 та RLE)»

Виконав:

Голубцов Дмитро ФІ-42мн

Перевірив:

Кандидат наук, доцент Яковлев С. В.

Зміст

1	Мета та завдання роботи	3
2	Архітектура програмного рішення	3
2.1	Основні патерни проектування	3
3	Опис програмних модулів та файлів	3
3.1	Загальні типи (common_types.h)	3
3.2	Модуль Base64	4
3.2.1	base64_strategy.h (Алгоритм кодування)	4
3.2.2	decoder_service.h (Валідація та потік)	4
3.3	Модуль RLE	4
3.3.1	rle_strategy.h (Алгоритм PackBits)	4
3.3.2	rle_service.h (Евристика кодування)	4
3.4	Точки входу (main_*.cpp)	4
4	Алгоритми роботи	5
5	Результати тестування	7
6	Висновок	7

1 Мета та завдання роботи

Мета роботи: Опанувати поняття коду, дослідити різні види перетворень даних, представлених двійковими словами (байтами), реалізувати алгоритми Base64 та RLE з використанням сучасних можливостей мови C++.

Завдання:

1. Реалізувати Base64-кодер та декодер з підтримкою опціональної довжини рядка (64 або 76 символів) та суворою валідацією вхідних даних згідно варіанту (українська локалізація помилок).
2. Реалізувати алгоритм стиснення RLE (Run-Length Encoding) згідно специфікації PackBits.
3. Забезпечити модульність архітектури та обробку помилок без використання виключень (exceptions).

2 Архітектура програмного рішення

Для виконання роботи було обрано архітектурний патерн **Strategy-Service**, який дозволяє чітко розділити алгоритмічну логіку (бітові операції) та бізнес-логіку програми (робота з файлами, валідація формату).

2.1 Основні патерни проектування

- **Strategy (Стратегія):** Використовується для інкапсуляції алгоритмів кодування/декодування. Класи `Base64Strategy` та `RleStrategy` є "чистими" (pure logic) — вони не залежать від файлової системи чи вводу-виводу, а працюють виключно з буферами даних.
- **Service (Сервіс):** Класи `EncoderService`, `DecoderService` та `RleService` відповідають за потік виконання: відкриття файлів, зчитування даних частинами, виклик стратегій та обробку помилок.
- **Monad (Result Pattern):** Замість механізму `try-catch` використовується клас-обгортка `Result<T>`, що дозволяє повертати або успішний результат, або структуровану помилку. Це підвищує надійність коду.

3 Опис програмних модулів та файлів

3.1 Загальні типи (`common_types.h`)

Цей файл визначає базові структури даних, що використовуються в усьому проекті:

- `struct AppError`: Структура для збереження інформації про помилку (номер рядка, позиція символу, повідомлення). Метод `to_string()` форматує повідомлення українською мовою згідно вимог лабораторної (наприклад, "Рядок 20, символ 02: Некоректний вхідний символ").
- `class Result<T>`: Реалізація монадичного типу, що містить або значення типу `T`, або `AppError`.
- `struct Config`: Зберігає налаштування запуску (шляхи до файлів, довжина рядка Base64).

3.2 Модуль Base64

3.2.1 base64_strategy.h (Алгоритм кодування)

Містить логіку побітових перетворень.

- **Генерація таблиць:** Використовує `constexpr` для створення масиву `DECODE_TABLE`, який дозволяє за $O(1)$ перетворити символ Base64 у числове значення.
- `encode_block`: Перетворює 3 вхідних байти (24 біти) у 4 символи Base64.
- `decode_block`: Перетворює 4 символи Base64 назад у 3 байти, враховуючи падінг ('=').

3.2.2 decoder_service.h (Валідація та потік)

Найскладніший модуль, оскільки реалізує суворі вимоги до формату вхідного файлу:

- **Ігнорування "фантомних" рядків:** Автоматично видаляє пробільні символи в кінці рядків, щоб уникнути помилкових попереджень.
- **Контроль довжини рядка:** Перевіряє, що всі рядки (крім останнього) мають фіксовану довжину (64 або 76 символів).
- **Детекція кінця повідомлення:** Відстежує момент, коли повідомлення логічно завершилось (через короткий рядок або падінг). Якщо після цього знайдено дані — видає попередження.

3.3 Модуль RLE

3.3.1 rle_strategy.h (Алгоритм PackBits)

Реалізує специфічні формули для RLE:

- **Repeat Header:** $128 + (length - 2)$ для послідовностей однакових байтів.
- **Literal Header:** $length - 1$ для послідовностей різних байтів.

3.3.2 rle_service.h (Евристика кодування)

В методі `encode` реалізовано алгоритм "жадібного" пошуку (Lookahead): 1. Програма сканує вхідний буфер. 2. Шукає послідовність однакових байтів довжиною ≥ 2 . 3. Якщо знайдено — записує Repeat Packet. 4. Якщо ні — накопичує Literal Packet, поки не зустрине початок нової повторюваної послідовності або не досягне ліміту (128 байт).

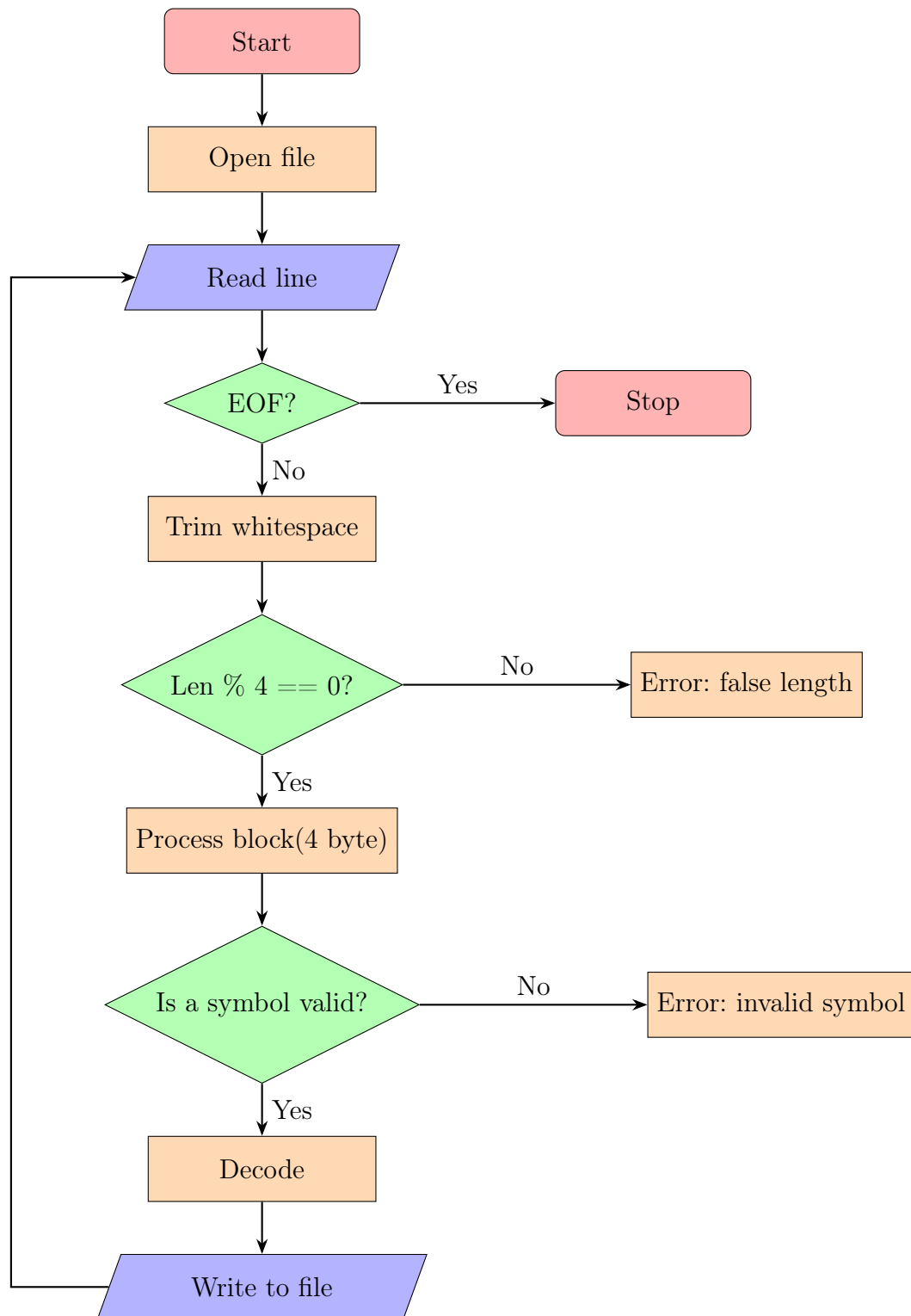
3.4 Точки входу (main_*.cpp)

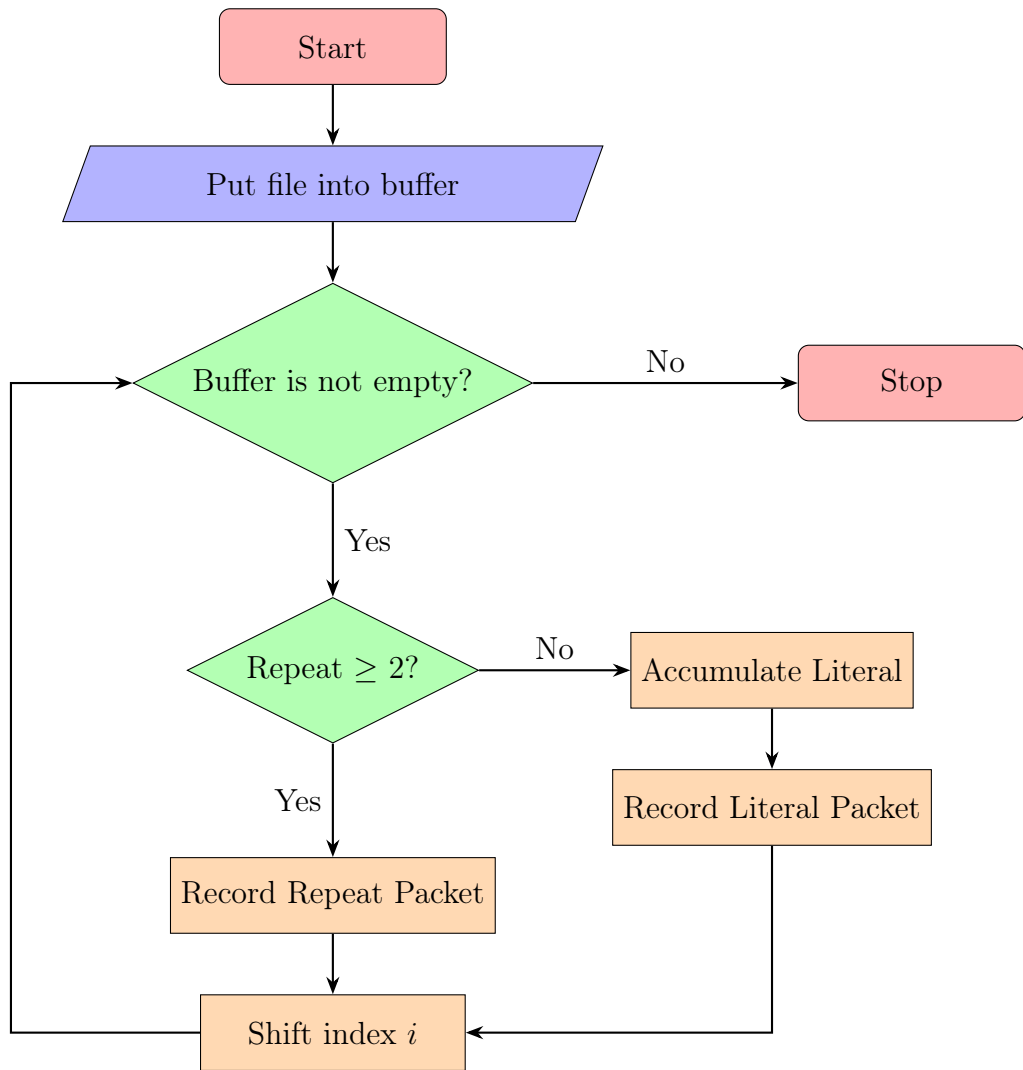
Чотири файли (`main_encode.cpp`, `main_decode.cpp`, `main_rle_encode.cpp`, `main_rle_decode.cpp`) відповідають за:

- Парсинг аргументів командного рядка (прапори `-s`, `-l`).
- Формування шляхів до файлів виводу (автоматичне додавання/видалення розширень).
- Виклик відповідного сервісу та вивід результату/помилки у консоль.

4 Алгоритми роботи

Нижче наведено детальні діаграми діяльності для ключових процесів декодування та кодування.





5 Результати тестування

Програма була протестована на наступних сценаріях:

1. Base64 Encoding:

- Вхід: Hello, World!
- Вихід: SGVsbG8sIFdvcmxkIQo=

2. Base64 Decoding (Strict):

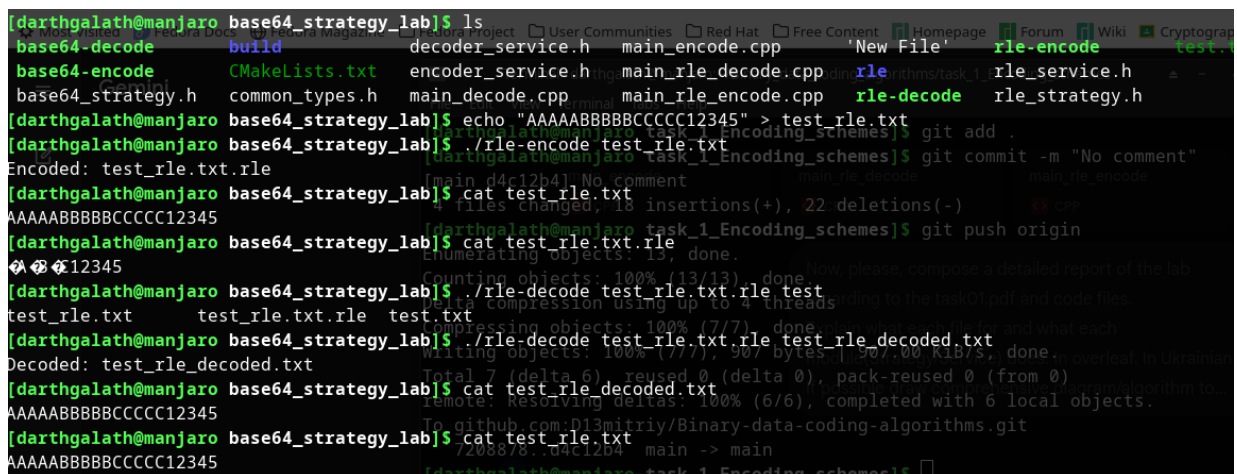
- Тест на невидимі пробіли: Декодер успішно ігнорує порожні рядки в кінці файлу, не видаючи помилкових попереджень.
- Тест на довжину: При зміні довжини рядка на 75 символів видається помилка "Рядок X: Некоректна довжина рядку (75)".

3. RLE Encoding:

- Вхід: AAAAABBBBBCCCCC12345
- Вихід: Стиснений файл містить заголовки 10000011 (для 5 'A'), 10000011 (для 5 'B') тощо.

6 Висновок

В ході лабораторної роботи було успішно реалізовано утиліти для кодування та декодування даних алгоритмами Base64 та RLE. Використання архітектури Strategy-Service дозволило створити гнучкий код, стійкий до змін вимог. Для імплементації застосовано стандарт C++20: `constexpr`, `std::format`.



```
[darthgalath@manjaro base64_strategy_lab]$ ls
base64-decode  build  decoder_service.h  main_encode.cpp  'New File'  rle_encode  test_rle.txt
base64-encode  CMakeLists.txt  encoder_service.h  main_rle_decode.cpp  rle  rle_service.h
base64_strategy.h  common_types.h  main_decode.cpp  main_rle_encode.cpp  rle-decode  rle_strategy.h

[darthgalath@manjaro base64_strategy_lab]$ echo "AAAAABBBBBCCCCC12345" > test_rle.txt
[darthgalath@manjaro base64_strategy_lab]$ ./rle-encode test_rle.txt
Encoded: test_rle.txt.rle
AAAAABBBBBCCCCC12345

[darthgalath@manjaro base64_strategy_lab]$ cat test_rle.txt.rle
AAAAABBBBBCCCCC12345

[darthgalath@manjaro base64_strategy_lab]$ ./rle-decode test_rle.txt.rle
Decoded: test_rle_decoded.txt
AAAAABBBBBCCCCC12345

[darthgalath@manjaro base64_strategy_lab]$ cat test_rle_decoded.txt
AAAAABBBBBCCCCC12345
```

Рис. 1:


```
[darthgalath@manjaro base64_strategy_lab]$ head -n 20 kali_setup.txt
Here's what it tells us:
Your radio supports monitor mode – confirmed by Supported interface modes listing * monitor.
That's the key requirement for wireless packet capture.
The interface is currently in AP mode (acting as an access point)!
The router's filesystem is extremely small: only ~320 KB writable space, with ~92 KB free.
That's not enough to install tcpdump or any large packages.
The ath9k driver on this hardware can do proper monitor mode and injection (it's the same chipset class as the Alfa AWUS036NHA).
So we'll use it purely as a remote capture radio, streaming live packets to Kali – no local storage or package installs needed.
Run these once:
sudo apt update
sudo apt install -y wireshark tcpdump
```

Рис. 4:

```
[darthgalath@manjaro base64_strategy_lab]$ cat test.txt
Hello, World!
[darthgalath@manjaro base64_strategy_lab]$ ./base64-encode test.txt
Encoded: test.txt.base64
[darthgalath@manjaro base64_strategy_lab]$ ls
base64-decode build decoder_service.h kali_setup.txt.base64 main_rle_decode.cpp rle-decode rle_strategy.h test.txt.base64
base64-encode CMakeLists.txt encoder_service.h main_decode.cpp main_rle_encode.cpp rle-encode test_rle.txt
base64_strategy.h common_types.h kali_setup.txt main_encode.cpp rle rle_service.h test.txt
[darthgalath@manjaro base64_strategy_lab]$ cat test.txt.base64
GVsbG8sIFdvcmxkIQo=
[darthgalath@manjaro base64_strategy_lab]$ ./rle-encode test.txt.base64
Encoded: test.txt.base64.rle
[darthgalath@manjaro base64_strategy_lab]$ cat test.txt.base64.rle
GVsbG8sIFdvcmxkIQo=
[darthgalath@manjaro base64_strategy_lab]$ ./base64-decode test.txt.base64.rle
Рядок 1: Некоректна довжина рядку (21)
[darthgalath@manjaro base64_strategy_lab]$ cp test.txt.base64.rle test_1.txt.base64
cp: overwrite 'test.txt.base64'? n
[darthgalath@manjaro base64_strategy_lab]$ cp test.txt.base64.rle test_1.txt.base64
[darthgalath@manjaro base64_strategy_lab]$ ./base64-decode test_1.txt.base64
Рядок 1: Некоректна довжина рядку (21)
[darthgalath@manjaro base64_strategy_lab]$ cat test_1.txt.base64
GVsbG8sIFdvcmxkIQo=
[darthgalath@manjaro base64_strategy_lab]$ stat -c %s test.txt.base64
21
[darthgalath@manjaro base64_strategy_lab]$ stat -c %s test_1.txt.base64
21
[darthgalath@manjaro base64_strategy_lab]$ ./rle-decode test_1.txt.base64 test_rle.txt
[darthgalath@manjaro base64_strategy_lab]$ ./rle-decode test_1.txt.base64
Decoded: test_1.txt.base64.unrle
[darthgalath@manjaro base64_strategy_lab]$
```

Рис. 5:

```
base64-encode CMakeLists.txt decoder_service.h main_decode.cpp main_rle_encode.cpp rle-encode test_rle.txt
base64_strategy.h common_types.h kali_setup.txt main_encode.cpp main_rle_decode.cpp rle-decode test.txt
[darthgalath@manjaro base64_strategy_lab]$ cat test.txt.base64
GVsbG8sIFdvcmxkIQo=
[darthgalath@manjaro base64_strategy_lab]$ ./rle-encode test.txt.base64
Encoded: test.txt.base64.rle
[darthgalath@manjaro base64_strategy_lab]$ cat test.txt.base64.rle
GVsbG8sIFdvcmxkIQo=
[darthgalath@manjaro base64_strategy_lab]$ ./base64-decode test.txt.base64.rle
Рядок 1: Некоректна довжина рядку (21)
[darthgalath@manjaro base64_strategy_lab]$ cp test.txt.base64.rle test.txt.base64
cp: overwrite 'test.txt.base64'? n
[darthgalath@manjaro base64_strategy_lab]$ cp test.txt.base64.rle test_1.txt.base64
[darthgalath@manjaro base64_strategy_lab]$ ./base64-decode test_1.txt.base64
Рядок 1: Некоректна довжина рядку (21)
[darthgalath@manjaro base64_strategy_lab]$ cat test_1.txt.base64
GVsbG8sIFdvcmxkIQo=
[darthgalath@manjaro base64_strategy_lab]$ stat -c %s test.txt.base64
21
[darthgalath@manjaro base64_strategy_lab]$ stat -c %s test_1.txt.base64
21
[darthgalath@manjaro base64_strategy_lab]$ ./rle-decode test_1.txt.base64 test_rle.txt
[darthgalath@manjaro base64_strategy_lab]$ ./rle-decode test_1.txt.base64
Decoded: test_1.txt.base64.unrle
[darthgalath@manjaro base64_strategy_lab]$ ls
base64-decode CMakeLists.txt kali_setup.txt main_rle_decode.cpp rle-encode test_1.txt.base64.unrle test.txt.base64.rle
base64-encode CMakeLists.txt kali_setup.txt.base64 main_rle_encode.cpp rle_service.h test_rle.txt
base64_strategy.h decoder_service.h main_decode.cpp rle rle_strategy.h test.txt
build CMakeLists.txt decoder_service.h encoder_service.h main_encode.cpp rle-decode test_1.txt.base64 test.txt.base64
[darthgalath@manjaro base64_strategy_lab]$ ./base64-decode test_1.txt.base64.unrle
Decoded: test_1.txt.base64.unrle.decoded
[darthgalath@manjaro base64_strategy_lab]$ cat test_1.txt.base64.unrle.decoded
Hello, World!
[darthgalath@manjaro base64_strategy_lab]$
```

Рис. 6: