C++ Module 09 - Exercise 00

Bitcoin Exchange - Final Implementation Plan

1. Project Structure

- Files: Makefile, main.cpp, BitcoinExchange.hpp, BitcoinExchange.cpp
- Program name: btc
- Compilation flags: -Wall -Wextra -Werror -std=c++98

2. Core Data Structure

- Use std::map to store exchange rates.
- Key format: YYYYMMDD converted to integer.
- Value: exchange rate (double).

3. Date Handling

- Validate format YYYY-MM-DD (length 10, correct positions for '-').
- Check valid month (1–12).
- Check valid day with leap year handling.
- Convert to integer key: y*10000 + m*100 + d.

4. Load Database (data.csv)

- Open file. If fail: print error and exit.
- Skip header line.
- Split each line by comma.
- Validate date and parse rate.
- Insert into map.

5. Input File Processing

- Check argc == 2. Else print error.
- Open input file. If fail: print error.
- Skip header if present.

- Split each line by '|'.
- Trim spaces.
- Validate date and value.

6. Value Validation

- Parse as double using istringstream.
- If parsing fails: Error: bad input => line
- If value < 0: Error: not a positive number.
- If value > 1000: Error: too large a number.

7. Finding Correct Rate

- Use lower_bound(dateKey).
- If exact match: use it.
- If not exact: use closest lower date.
- If earlier than first DB date: use earliest available.

8. Output Format

- Format: YYYY-MM-DD => value = result
- Result = value * exchange_rate
- No extra text, no colors.

9. Required Helpers

- trim(string)
- parseDate(string, int&)
- daysInMonth(int y, int m)
- isLeapYear(int y)

10. Testing Checklist

- Run without argument -> error.
- Invalid date -> Error: bad input => ...
- Negative value -> Error: not a positive number.
- Too large value -> Error: too large a number.
- Missing date in DB -> uses closest lower date.