COMPUTER ENGINEERING WORKSHOP

S.E. (CIS) OEL REPORT

Project Group ID:

M.UMER AHMED	CS-095
MALIK HASHIR SHAKEEL	CS- 096
M.HASSIN SAGHEER	CS-129

BATCH: 2023

DEPARTMENT OF COMPUTER AND INFORMATION SYSTEMS ENGINEERING

NED University of Engineering. & Tech.,
Karachi-75270

CONTENTS

CHAPTER 1

PROBLEM DESCRIPTION

The goal of this project is to develop a weather application that can fetch real-time weather data for a specified city using an API. The application processes, stores, and displays this data in a user-friendly manner. It also maintains historical data, calculates averages, and provides alerts for extreme weather conditions.

KEY FEATURES

1.**Weather Data Fetching**

- The application retrieves real-time weather data for a city specified by the user via a shell script.

2. **Data Storage**

- **Raw Data**: Saved in JSON format to maintain a historical record of the original API responses.

- **Processed Data** Saved in a text file with a well-formatted layout for easy readability.

3. **Alerts**

- Sends popup alerts using **Zenity** when the temperature crosses predefined thresholds (high or low).

4. **Average Calculation**

- Computes and displays the average temperature for the last 24 readings (or multiples of 24).

5. **Modular Design**

- The program is divided into three files:

  - `**main.c**` for the program logic.

  - `**functions.c**` for reusable functions.

  - `**functions.h**` for function prototypes and data structure definitions.

6. **Automation**

- A `**Makefile**` automates the build process, while a shell script automates running the program at regular intervals using cron jobs.

7. **User Interface**

- Prints the current temperature and condition in a readable format to the terminal.

8. **Ease of Use**

- The city name is passed as an argument to the shell script, allowing flexibility and simplicity for the user.

This program combines real-time data fetching, storage, and user alerts, making it a comprehensive solution for monitoring weather conditions.

CHAPTER 2

METHODOLOGY FOR DEVELOPING THE WEATHER APPLICATION

1. Requirement Analysis

- **Objective:** Identify core functionalities and features.

- **Key Requirements:**

1. Fetch weather data for a user-specified city using an API.
2. Store data in two formats: raw data (JSON) and processed data (text).
3. Maintain historical records for both raw and processed data.
4. Provide alerts for extreme temperatures using Zenity pop-ups.
5. Display the current temperature and condition in a clear format.
6. Compute and display the average temperature after 24 or multiples of 24 readings.
7. Allow the city name to be passed dynamically via a shell script.
8. Ensure maintainability with a modular structure (`**main.c**`, `**functions.c**`, `**functions.h**`).

2. System Design

- **Modular Architecture**

- **`main.c`:** Manages the overall flow of the program, including input handling, data fetching, and user alerts.

- **`functions.c`:** Implements reusable functions for data fetching, processing, storage, and calculations.

- **`functions.h`:** Contains function declarations and the `WeatherData` structure.

- **Automated Workflow:**

  - A **`Makefile`** automates the compilation, linking, and cleaning processes.

  - A shell script **(`automation.sh`)** allows periodic execution via **cron** jobs.

3. Implementation

a. City Name Input

  - Used the command-line argument **(`argv[1]`)** to accept the city name.

  - Implemented a formatting function to capitalize the first letter and lowercase the rest of the city name.

b. Fetching Weather Data

  - Used **`libcurl`** to send API requests to **OpenWeather.**

  - Saved the API response in a raw data file **(`raw_data.json`).**

c. Data Processing

  - Parsed the JSON response using **`libcjson`** to extract the temperature and weather conditions.

  - Stored the processed data in a human-readable text file **(`processed_data_history.txt`).**

d. Historical Data

- Appended each API response to **`raw_data_history.json`.**

- Maintained a cumulative record of processed data for analysis.

e. Alerts

- Integrated **Zenity** pop-ups to alert users about:

  - High temperature (>30°C).

  - Low temperature (<10°C).

f. Average Calculation

- Checked if the processed data file had 24 or a multiple of 24 readings.

- Calculated the average temperature and displayed it on the terminal.

- Logged the calculated average into **`average_temperatures.txt`.**

g. Automation

- Developed a shell script to run the application periodically.

- Added **cron** job compatibility for executing the shell script at regular intervals.

4. Testing

- **Data Accuracy:**

  - Verified API responses were fetched correctly.

  - Checked JSON parsing for accurate extraction of temperature and conditions.

- **Alert Functionality:**

  - Simulated extreme weather conditions to ensure alerts triggered as expected.

- **Automation:**

  - Tested the shell script for passing city names dynamically and running the program at scheduled times.

- **File Management:**

  - Ensured raw and processed data histories were maintained correctly.

5. Deployment

- Packaged the application with:

  - Source files: **`main.c`, `functions.c`, `functions.h`.**

  - A **`Makefile`** for easy compilation.

  - A shell script **(`automation.sh`)** for automation.

CHAPTER 3

RESULTS

The weather application successfully meets the functional requirements, providing accurate and timely weather data while maintaining historical records.

KEY RESULTS

1. Real-Time Data Fetching

- Retrieves accurate weather data for the specified city using the **OpenWeather** API.

2. Data Storage

- Raw JSON responses saved in **`raw_data.json`** and appended to **`raw_data_history.json`.**

- Processed data (temperature and conditions) saved in `processed_data_history.txt`.

3. Alerts

- **High Temperature (>30°C): Zenity** pop-ups notify the user.

- **Low Temperature (<10°C):** Pop-ups are triggered for cold weather.

4. Average Temperature Calculation

- Displays averages for every 24 or multiple of 24 readings.

- Logs averages into **`average_temperatures.txt`.**

5. Current Weather Display

- Shows current temperature and weather conditions in a formatted terminal output.

6. Automation

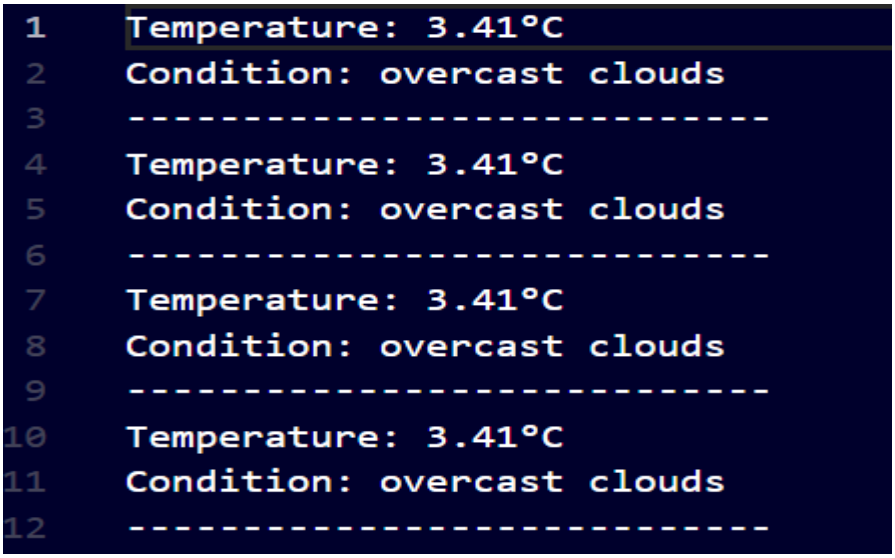
- Supports execution via shell script with **cron** compatibility for periodic updates.

OUTPUT SCREENSHOTS

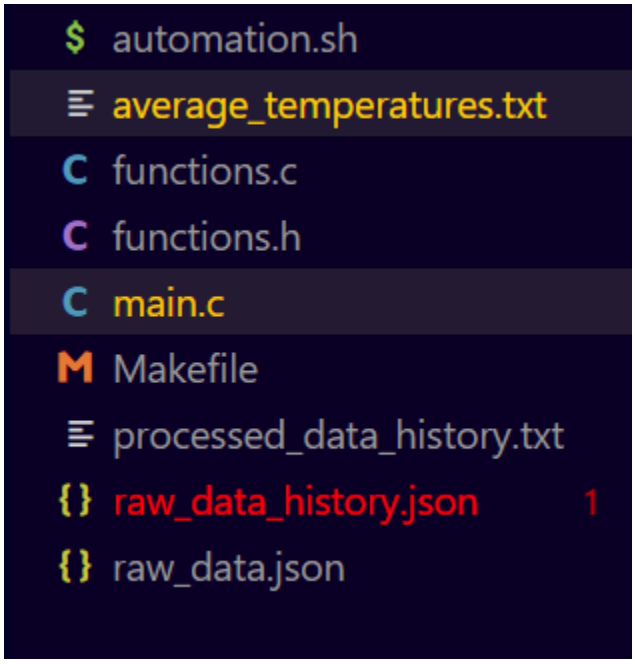
- Terminal Output of current weather along with **zenity** alerts.

- Screenshot of the **raw_data_history.json** showing a portion of the fetched raw API response.

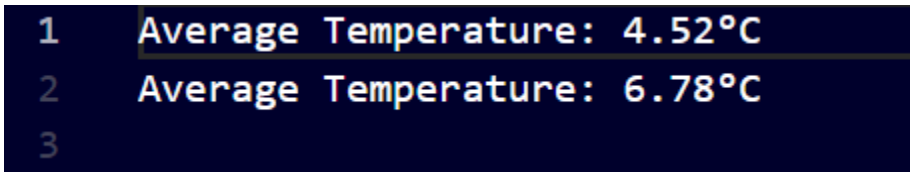
- Screenshot of **processed_data_history.txt** showing the formatted weather data.



- Screenshot showing the directory structure where files like **main.c, functions.c, functions.h, Makefile,** and data files **(raw_data.json, processed_data_history.txt, etc.)** are stored.



- Screenshot of the **average_temperature.txt** output showing the calculated average temperature when 24 readings are reached.



CONCLUSION

The application is functional, user-friendly, and modular, ensuring accurate weather monitoring with timely alerts and historical tracking.