# COMPUTER ENGINEERING WORKSHOP

**S.E. (CIS) OEL REPORT**

**Project Group ID:**

**NAME OF MEMBER #1**

MUHAMMAD SAAD - CS-23148

**NAME OF MEMBER #2**

MUHAMMAD ARIF MAHDI - CS-23057

**NAME OF MEMBER #3**

ZAIN JAVED – CS-23060

**BATCH:** 2023

**Department of Computer and Information Systems Engineering**

**NED University of Engg. & Tech.,**

**Karachi-75270**

**CONTENTS**

**S.No. Page No.**

1. **Problem Description** 1

1. **Methodology**  2-3

1. **Results**  4-5

**CHAPTER 1:**

# PROBLEM DESCRIPTION

In an era of increasing environmental concerns, real-time monitoring of key environmental parameters has become essential for better awareness and timely responses. This project focuses on building an **Integrated Environmental Monitoring System** that utilizes the capabilities of C programming, Linux shell scripting, and external APIs. It allows for the efficient retrieval and logging of real-time environmental data while providing automated alerts under critical conditions.

**Objectives:**

1. **Data Integration**: Leverage APIs to collect real-time data, such as temperature and humidity.
2. **Dynamic Alerts**: Notify personnel about critical environmental readings via terminal, syslog, or email.
3. **File Management**: Log environmental data in a structured and easily readable format.
4. **Automation**: Automate the entire monitoring process through shell scripts and cron jobs.
5. **Optimized Code Design**: Incorporate pointers, dynamic memory allocation, and header files to ensure resource efficiency and modular code structure.
6. **Scalability**: Lay the foundation for adding future functionalities, such as wind speed and air quality monitoring.

So we are required to develop a system that can address the challenges of real-time monitoring by offering reliable, data retrieval, Automation, and user decided Alerts.

We are using the Linux system for the entire program making, as it would help us greatly in generating real time alerts and other features like Automation etc.

**CHAPTER 2:**

**METHODOLOGY**

**1. API Integration**

The system interacts with the OpenWeather API to fetch real-time weather data. The **CURL** library was used to make HTTP GET requests, retrieving environmental data in JSON format.  
Key parameters retrieved include:

* **City and Country:** Identify the geographical location of the data.
* **Temperature (in Celsius):** Real-time temperature data.
* **Humidity (in %):** Percentage of air moisture.

**2. Data Processing and Parsing**

* The JSON-C library was used to parse the JSON data received from the API.
* Relevant fields (e.g., city name, country code, temperature, humidity) were extracted and stored in a structured manner.
* A custom-defined struct (WeatherData) was employed to organize the parsed data, ensuring code clarity and scalability

**3. Data Logging**

Processed data is appended to a file named weather\_data.txt.

Example log entry:

**City: Karachi, Country: PK, Temperature: 24.90°C, Humidity: 65%**

**4. Alerts**

**Terminal Alerts:**

* Each iteration of the program generates a terminal beep to alert the user of new environmental data retrieval.

**Critical Notifications:**

* If the temperature exceeds 30°C or if the humidity drops below 20%, critical notifications are generated using **syslog**.
* These notifications are logged with appropriate severity levels (LOG\_ALERT for temperature and LOG\_WARNING for humidity).
* The syslog-based alerts are seamlessly integrated into the Linux environment, ensuring system-level visibility and enabling further integration with tools like desktop notification.

**Desktop Notifications via Syslog:**

* On compatible Linux environments, the syslog alerts, we have used to trigger desktop notifications to provide a graphical alert interface for critical conditions.
* This enhances user awareness and ensures prompt attention to significant environmental changes.
* This is helping us keep the track of critical environmental conditions and notify us as the conditions become critical.

**5. Automation**

The monitoring process was automated using a shell script and scheduled with a cron job:

* A shell script runs the weather-fetching program every 60 seconds, appends the output to the log file, and generates a beep alert.
* Cron was configured to execute the script continuously, ensuring uninterrupted automation.

**6. Optimization**

* **Dynamic Memory Allocation**: Used to efficiently allocate memory for API responses and weather data structs.
* **Header Files**: Encapsulated reusable components, enhancing code readability and maintainability.

**7. Challenges Encountered**

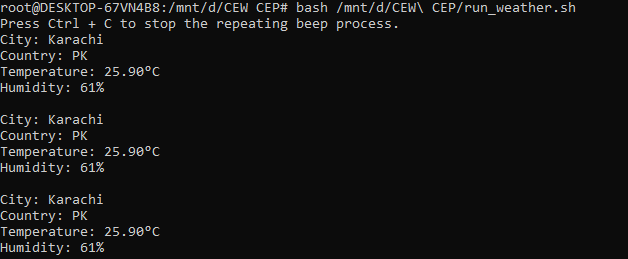
* Configuring Sendmail for email alerts was challenging due to authentication and domain issues, although we moved to implementing desktop notification alert.
* D-Bus constraints in a non-GUI environment prevented desktop notifications as we were first using Ubuntu Terminal (only) for implications and its non-GUI environment.
* Debugging cron job behavior required careful attention to environment variables.

**CHAPTER 3:**

**RESULTS**

**System Achievements:**

1. **Successful Data Retrieval**:
   * The system successfully fetched data from OpenWeather API and displayed it in the terminal.
   * Example Output:



* Also the system successfully saves the output data in the (weather data) file.

1. **Real-Time Terminal Alerts**:
   * Beep sound alerts were generated after every data retrieval, ensuring attention during updates.
2. **Automated Logging**:
   * Environmental data was logged into weather\_data.txt at one-minute intervals using a cron job, along with displaying it on the output terminal, also we have (optionally) implemented the automation using a while loop on program in the (.sh) bash file (run\_weather.sh), as now the program could automate upon running the terminal every time or if we enable our cronjob It would then automate for long-term where no user interaction is required, independent of manual intervention.
3. **Critical Notifications**:
   * Alerts for high temperature or low humidity were logged dynamically in the main file (weather-fetch.c)
4. **Scalable Design**:
   * The system was modular and ready for extension, such as including additional environmental parameters like wind speed or pressure.

**Challenges:**

* Configuring Sendmail for email alerts posed issues due to domain-related restrictions.
* D-Bus limitations prevented desktop notifications in a non-GUI environment, which we were first implementing on (Ubuntu Terminal), and then we moved to Linux VM, having a GUI-environment, to resolve the problem.

**CHAPTER 4: CONCLUSION**

The **Integrated Environmental Monitoring System** demonstrated the ability to retrieve, process, and log environmental data in real-time. While email alerts require further configuration, the system successfully met most objectives and showcased robust automation and modularity using C and Linux. Future improvements can include GUI notifications and more comprehensive alert mechanisms.

**Future Improvements:**

1. **Email Alerts**: Email alerts can also be added using Sendmail feature.
2. **Extended Data**: Include parameters like wind speed, air quality index, and precipitation data.
3. **Advanced Automation**: Leverage Python for more complex scheduling and error handling.

The project serves as a practical demonstration of applying programming and automation to solve real-world problems effectively. However, it leads every group member of ours to learn new and skillful operating skills, like system management, Linux-usage and command, Api-usage, C language implications, bash file usage, alert-generating and mainly Automation. Although, each member contributed to the development process, handling different aspects of the program, and in the end, the combined efforts led to the successful implementation and results.