# COMPUTER ENGINEERING WORKSHOP

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

**Project Group ID:**


| | |
|---|---|
| Muhammad Qambar Hussain | CS-23130 |
| Asadullah Nizami | CS-23092 |
| Ali Raza Baloch | CS-23130 |


**BATCH:** 2023


**Department of Computer and Information Systems Engineering**

**NED University of Engg. & Tech.,**
**Karachi-75270**


**CONTENTS**

# Contents

**DEPARTMENT OF COMPUTER & INFORMATION SYSTEMS ENGINEERING**
**BACHELORS IN COMPUTER SYSTEMS ENGINEERING**
**Course Code: CS-219**
**Course Title: Computer Engineering Workshop**
**Open Ended Lab**
**SE Batch 2023, Fall Semester 2024**
**Grading Rubric**
**TERM PROJECT**

**Group Members:**

| Student No. | Name | Roll No. |
|---|---|---|
| S1 | **Muhammad Qambar Hussain** | **CS-23125** |
| S2 | **Asadullah Nizami** | **CS-23092** |
| S3 | **Ali Raza Baloch** | **CS-23130** |

| **CRITERIA AND SCALES** | | | | **Marks Obtained** | | |
|---|---|---|---|---|---|---|
| | | | | **S1** | **S2** | **S3** |
| Criterion1: Has the student implemented an efficient and scalable solution for data retrieval, processing, and reporting? | | | | | | |
| 0 | 1 | 2 | 3 | | | |
| The student has not even implemented a basic solution that meets the project's requirements. | The student has implemented a basic solution that meets the project's requirements but may lack optimization in certain aspects. | The student has implemented a proficient and well-optimized solution. | The student has implemented an exceptionally efficient and scalable solution. | | | |
| Criterion 2: Has student demonstrated a strong understanding of C programming fundamentals? | | | | | | |
| 0 | 1 | 2 | 3 | | | |
| The student doesn't have basic understanding of C programming fundamentals. | The student exhibits a basic understanding of C programming fundamentals. | The student demonstrates a strong understanding of C programming fundamentals. | The student demonstrates an exceptional understanding of C programming fundamentals. | | | |
| Criterion 3: How well written is the report? | | | | | | |
| 0 | 1 | 2 | 3 | | | |
| The submitted report is unfit to be graded. | The report is partially acceptable. | The report is complete and concise. | The report is exceptionally written. | | | |
| | | | Total Marks: | | | |

# PROBLEM DESCRIPTION

An integrated environmental monitoring system in C, focused on practical applications and efficient programming techniques. The system interacts with a free API to gather real-time environmental data, and key objectives include: • Fetching real-time environmental data (e.g., temperature, humidity) by interfacing with a free API.

• Saving both raw and processed data into files for storage and future analysis.

• Developing shell scripts to automate the tasks of retrieving and processing environmental data.

• Optimizing data handling using pointers and dynamic memory allocation to boost performance.

• Setting up real-time notifications using Linux system calls to alert staff of critical environmental conditions.

• Organizing the code into separate header files, promoting modularity and clarity for improved readability and maintenance

# METHODOLOGY

## Requirement Analysis

- Key features were identified: real-time retrieval, storage, processing, and reporting of environmental data.
- A free API (**OpenWeatherMap**) was selected to provide the necessary current environmental data.
- The scope was defined, and the tools required were chosen: C programming(**Clion Editor**), shell scripting(**Nano Editor**), and a Linux environment(**Ubuntu and WSL 2**).

## System Design

- **Architecture Planning**:
  - The system was divided into functional modules such as weather data retrieval, processing, alerts, and automation.
- **Data Flow Design**:
  - The flow of data from the API to storage, processing, and output stages was outlined.
- **File Structure**:
  - Formats for raw and processed data storage were determined, such as **.csv** and **.txt** files.

## Implementation

## API Interaction:

- **CURL** utility was used to fetch real-time environmental data via **HTTP** requests.
- JSON responses from the API were parsed using **CJSON** library to extract relevant data (temperature, humidity).


## Data Storage:

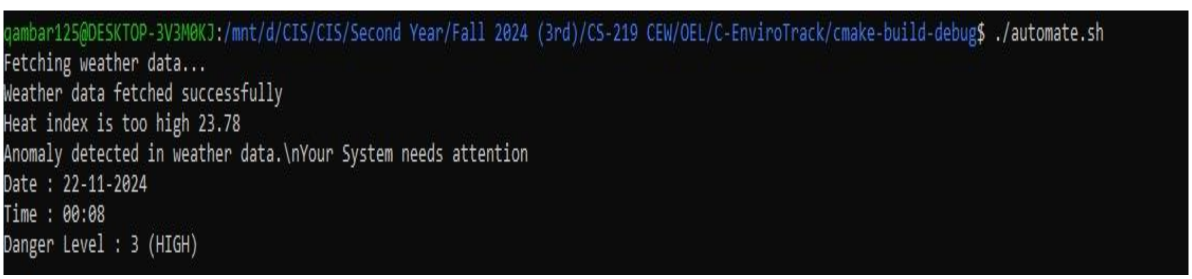
- File handling in C was implemented to store raw data retrieved from the API.
- Mechanisms for writing processed data to files were created.

## Data Processing:

- Mathematical operations were applied to process the data like calculating **Heat Index** and **Dew Point**.
- Dynamic memory allocation(**`malloc, realloc and free`**) and pointers were used to optimize data handling and efficiency.

## Real-Time Alerts:

- Linux system calls (signals or notifications) were utilized to trigger alerts when critical environmental thresholds were exceeded.



## Task Scheduling and Shell Scripting:

- Added entry(job) for Cron tab file

```
MAILTO=""
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
* * * * * /mnt/d/CIS/CIS/Second Year/Fall 2024 (3rd)/CS-219 CEW/OEL/C-EnviroTrack/cmake-build-debug/automate.sh cron_log.txt 2>&1
```

- Shell scripts (**automate.sh** and **sendAlert.sh**) were developed to automate tasks such as periodic data retrieval, processing, triggering alerts, file clean up, and log updates.

```bash
#!/bin/bash

# Script to execute main.c every hour to fetch weather data

# Path to compiled executable of C_EnviroTrack.c
MAIN_EXECUTABLE="./C_EnviroTrack.exe"

# Check if the main executable exists
if [ ! -f "$MAIN_EXECUTABLE" ]; then
    echo "Error: $MAIN_EXECUTABLE not found. Please compile main.c first."
    exit 1
fi

# Infinite loop to run the executable every hour
while true; do
    # Run the main executable
    $MAIN_EXECUTABLE

    # Wait for one hour before the next execution
   sleep 3600
done
```

```bash
#!/bin/bash

DANGER_LEVEL=3
LEVEL_DESCRIPTION="HIGH"
DATE=$(date +"%d-%m-%Y")
TIME=$(date +"%H:%M")
echo "Anomaly detected in weather data.\nYour System needs attention"
echo "Date : $DATE"
echo "Time : $TIME"
echo "Danger Level : $DANGER_LEVEL ($LEVEL_DESCRIPTION)"


## Email configuration
#TO="<Email>@gmail.com"
#SUBJECT="***Anomaly in weather***"
#BODY="An anomaly is detected in the weather data at:\n\nDate: $DATE\nTime: $TIME\n\n        ***ANOMALY***        \n\nDanger Level: $DANGER_LEVEL\n\nPlease find the
#ATTACHMENT_FILE="<Add path of process.csv>"
#
## Path to the password file
#PASSWORD_FILE="password.txt"
#
## Check if the password file exists
#if [ ! -f "$PASSWORD_FILE" ]; then
#    echo "Error: Password file not found. Please create 'password.txt' containing the SMTP password."
#    exit 1
#fi
#
## Read the SMTP password from the file
#SMTP_PASSWORD=$(cat "$PASSWORD_FILE")
#
## Send the email using sendemail
#sendemail -f "<Email>@gmail.com" \
#          -t "$TO" \
#          -u "$SUBJECT" \
#          -m "$BODY" \
#          -a "$ATTACHMENT_FILE" \
#          -s "smtp.gmail.com:587" \
#          -o tls=yes \
#          -xu "<Email>@gmail.com" \
                                        [ Read 41 lines ]
```

## Testing and Validation

- The system was tested for API connectivity and data retrieval accuracy.
- Data processing logic and file storage were validated to ensure correctness.
- Edge cases (e.g., invalid API responses, extreme environmental readings) were simulated to ensure robust behaviour.
- Real-time alert mechanisms were tested, and the functionality of automation scripts was verified.

## Integration and Deployment

- All modules (API interaction, data processing, and alert system) were integrated into a cohesive system.
- The system was deployed in a Linux environment, ensuring that all scripts and programs ran smoothly.
- **Header files** were made for every corresponding C files to store function definitions.

## Documentation and Reporting

- The code was documented with comments and organized using header files for modularization.
- A **`README.md`** file was created to outline all essential aspects of the project, including system requirements, setup instructions, and usage guidelines.

# RESULT

## Performance Evaluation

- **Data Retrieval:** Verified the correctness of the data fetched from the API under different network conditions.
- **Processing Efficiency:** Processing speed optimized through dynamic memory allocation and pointer usage.

main.c | assign_level_and_trigger.c | sendAlert.sh | weather_data.csv | automate.sh | raw_response.txt

| TEMP | HUMIDITY | PRESSURE | HEAT_INDEX | DEW_POINT | DANGER |
|---|---|---|---|---|---|
| 297.05 | 69.00 | 1014.00 | 24.88 | 17.86 | HIGH |
| 297.05 | 69.00 | 1014.00 | 24.88 | 17.86 | HIGH |
| 297.05 | 69.00 | 1014.00 | 24.88 | 17.86 | HIGH |
| 297.05 | 69.00 | 1014.00 | 24.88 | 17.86 | HIGH |
| 296.05 | 73.00 | 1013.00 | 23.78 | 17.80 | HIGH |
| 296.05 | 73.00 | 1013.00 | 23.78 | 17.80 | HIGH |
| 296.05 | 73.00 | 1013.00 | 23.78 | 17.80 | HIGH |
| 296.05 | 73.00 | 1013.00 | 23.78 | 17.80 | HIGH |

## Challenges Overcome

- **Secure handling of API key:** Function to retrieve API key from **.env file**

```c
// Function to read the API key from the .env file
char* get_api_key(const char* file_path) {
    FILE* file = fopen( Filename: file_path,  Mode: "r");
    if (!file) {
        perror( ErrMsg: "Error opening .env file");
        return NULL;
    }

    char line[256];
    char* api_key = NULL;

    while (fgets(line,  MaxCount: sizeof(line), file)) {
        if (sscanf( source: line,  format: "API_KEY=%ms", &api_key) == 1) {
            break; // API key successfully extracted
        }
    }

    fclose(file);
    return api_key;
}
```

- **Error Handling and Debugging:** It is used for debugging purposes i.e. it creates a errorLog.txt file that stores list of errors with their time stamps.

```c
void logError(const char *message, const char *error) {

    //if there is no "error" message it will print NULL
    FILE *logFile = fopen("errorLog.txt", "a+");

    if (logFile != NULL) {
        time_t t = time(NULL);
        struct tm *tm_info = localtime(&t);

        char timestamp[20];
        strftime(timestamp, sizeof(timestamp), "%Y-%m-%d %H:%M:%S", tm_info);

        fprintf(logFile, "[%s] %s: %s\n", timestamp, message, error);
        fclose(logFile);
    }
}
```

## Output Samples

- Bash Scripting and fetching data every hour

```
qambar125@DESKTOP-3V3M0KJ:/mnt/d/CIS/CIS/Second Year/Fall 2024 (3rd)/CS-219 CEW/OEL/C-EnviroTrack/cmake-build-debug$ ./automate.sh
Fetching weather data...
Weather data fetched successfully
Heat index is too high 23.78
Anomaly detected in weather data.\nYour System needs attention
Date : 22-11-2024
Time : 00:08
Danger Level : 3 (HIGH)
```