DEP Internship Batch 2

C++ Programming

Rida Zahra

EME, NUST

Task 1

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

class AirQualityForecastingSystem {
private:
    vector<string> airQualityData; // Placeholder for air quality data

public:
    void fetchAirQualityData() {
        // Placeholder for API call
        airQualityData.clear();
        airQualityData.push_back("Location: City A, PM2.5: 35 µg/m³, PM10: 50 µg/m³");
        airQualityData.push_back("Location: City B, PM2.5: 40 µg/m³, PM10: 55 µg/m³");
        cout << "Air quality data fetched successfully!" << endl;
    }

    void displayAirQualityData() {
        if (airQualityData.empty()) {
            cout << "No air quality data available." << endl;
            return;
        }
        cout << "Air Quality Data:" << endl;
        for (const auto& data : airQualityData) {
```

```
            cout << data << endl;
        }
    }
};

int main() {
    AirQualityForecastingSystem system;
    int choice;

    while (true) {
        cout << "1. Fetch Air Quality Data" << endl;
        cout << "2. Display Air Quality Data" << endl;
        cout << "3. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;
        cin.ignore();  // To ignore the newline character after entering choice

        switch (choice) {
            case 1:
                system.fetchAirQualityData();
                break;
            case 2:
                system.displayAirQualityData();
                break;
            case 3:
                cout << "Exiting..." << endl;
                return 0;
            default:
                cout << "Invalid choice. Please try again." << endl;
```

```
            break;
        }
    }
}
```

—---

Pseudo Code for Advanced Implementation

```
class AirQualityForecastingSystem {
private:
    vector<string> airQualityData;
    bool isOnline; // Track if the system is online or offline

public:
    void fetchAirQualityData() {
        if (isOnline) {
            // Fetch data from cloud
            // Example: Fetch data from API and store in airQualityData
        } else {
            // Load data from local storage
            // Example: Read from a local file or database
        }
        cout << "Air quality data fetched successfully!" << endl;
    }

    void saveDataToCloud() {
        if (isOnline) {
            // Save airQualityData to Google Firestore
            // Example: Use Firestore API to upload data
```

```
    } else {
        cout << "Cannot save data in offline mode." << endl;
    }
}

void saveDataLocally() {
    // Save airQualityData to local storage
    // Example: Write to a file or local database
}

void retrieveDataFromCloud() {
    if (isOnline) {
        // Retrieve data from Google Firestore
        // Example: Use Firestore API to fetch data
    } else {
        cout << "Cannot retrieve data in offline mode." << endl;
    }
}

void retrieveDataLocally() {
    // Retrieve data from local storage
    // Example: Read from a file or local database
}

void displayAirQualityData() {
    if (airQualityData.empty()) {
        cout << "No air quality data available." << endl;
        return;
    }
```

```
        cout << "Air Quality Data:" << endl;
        for (const auto& data : airQualityData) {
            cout << data << endl;
        }
    }
};
```

Task 2

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

class Contact {
public:
    string name;
    string phoneNumber;

    Contact(string name, string phoneNumber) : name(name), phoneNumber(phoneNumber)
{}
};

class ContactManager {
private:
    vector<Contact> contacts;

public:
    void addContact(string name, string phoneNumber) {
        contacts.push_back(Contact(name, phoneNumber));
        cout << "Contact added successfully!" << endl;
    }

    void viewContacts() {
```

```
        if (contacts.empty()) {
            cout << "No contacts available." << endl;
            return;
        }

        cout << "Contacts List:" << endl;
        for (size_t i = 0; i < contacts.size(); ++i) {
            cout << "Name: " << contacts[i].name << ", Phone Number: " <<
contacts[i].phoneNumber << endl;
        }
    }

    void deleteContact(string name) {
        for (auto it = contacts.begin(); it != contacts.end(); ++it) {
            if (it->name == name) {
                contacts.erase(it);
                cout << "Contact deleted successfully!" << endl;
                return;
            }
        }
        cout << "Contact not found." << endl;
    }
};

void displayMenu() {
    cout << "Contact Management System" << endl;
    cout << "1. Add Contact" << endl;
    cout << "2. View Contacts" << endl;
    cout << "3. Delete Contact" << endl;
```

```
    cout << "4. Exit" << endl;
}

int main() {
    ContactManager manager;
    int choice;
    string name, phoneNumber;

    while (true) {
        displayMenu();
        cout << "Enter your choice: ";
        cin >> choice;
        cin.ignore();  // To ignore the newline character after entering choice

        switch (choice) {
            case 1:
                cout << "Enter name: ";
                getline(cin, name);
                cout << "Enter phone number: ";
                getline(cin, phoneNumber);
                manager.addContact(name, phoneNumber);
                break;

            case 2:
                manager.viewContacts();
                break;

            case 3:
                cout << "Enter name of contact to delete: ";
```

```
            getline(cin, name);
            manager.deleteContact(name);
            break;

        case 4:
            cout << "Exiting..." << endl;
            return 0;

        default:
            cout << "Invalid choice. Please try again." << endl;
            break;
    }
  }
}
```

Task 3

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>

using namespace std;

// Function to compress the content of a file using Run-Length Encoding
void compressFile(const string& inputFileName, const string& outputFileName) {
    ifstream inputFile(inputFileName, ios::binary);
    ofstream outputFile(outputFileName, ios::binary);

    if (!inputFile.is_open() || !outputFile.is_open()) {
        cerr << "Error opening file!" << endl;
        return;
    }

    char currentChar;
    char previousChar = '\0';
    int count = 0;

    while (inputFile.get(currentChar)) {
        if (currentChar == previousChar) {
            count++;
        } else {
            if (count > 0) {
```

```
            outputFile.put(previousChar);
            outputFile.put(static_cast<char>(count));
        }
        previousChar = currentChar;
        count = 1;
    }
}

// Write the last run
if (count > 0) {
    outputFile.put(previousChar);
    outputFile.put(static_cast<char>(count));
}

inputFile.close();
outputFile.close();
cout << "Compression completed." << endl;
}

// Function to decompress the content of a file using Run-Length Encoding
void decompressFile(const string& inputFileName, const string& outputFileName) {
    ifstream inputFile(inputFileName, ios::binary);
    ofstream outputFile(outputFileName, ios::binary);

    if (!inputFile.is_open() || !outputFile.is_open()) {
        cerr << "Error opening file!" << endl;
        return;
    }
```

```
    char currentChar;
    char count;

    while (inputFile.get(currentChar)) {
        inputFile.get(count);
        for (int i = 0; i < static_cast<unsigned char>(count); ++i) {
            outputFile.put(currentChar);
        }
    }

    inputFile.close();
    outputFile.close();
    cout << "Decompression completed." << endl;
}

int main() {
    int choice;
    string inputFileName, outputFileName;

    cout << "1. Compress File" << endl;
    cout << "2. Decompress File" << endl;
    cout << "3. Exit" << endl;
    cout << "Enter your choice: ";
    cin >> choice;
    cin.ignore();  // To ignore the newline character after entering choice

    switch (choice) {
        case 1:
            cout << "Enter the input file name: ";
```

```
            getline(cin, inputFileName);
            cout << "Enter the output file name: ";
            getline(cin, outputFileName);
            compressFile(inputFileName, outputFileName);
            break;

        case 2:
            cout << "Enter the input file name: ";
            getline(cin, inputFileName);
            cout << "Enter the output file name: ";
            getline(cin, outputFileName);
            decompressFile(inputFileName, outputFileName);
            break;

        case 3:
            cout << "Exiting..." << endl;
            return 0;

        default:
            cout << "Invalid choice. Please try again." << endl;
            break;
    }

    return 0;
}
```

Task 4

1. Setting Up Socket Programming

```
#include <iostream>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>

#define PORT 8080

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);

    // Create socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    // Forcefully attach socket to the port 8080
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt,
sizeof(opt))) {
        perror("setsockopt");
        exit(EXIT_FAILURE);
```

```
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    // Bind the socket to the port
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(server_fd, 3) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    }

    std::cout << "Server is listening on port " << PORT << std::endl;

    while (true) {
        if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
(socklen_t*)&addrlen)) < 0) {
            perror("accept");
            exit(EXIT_FAILURE);
        }

        // Handle the client connection in a new thread
    }
```

```
    return 0;
}
```

2. Implementing Multi-Threading

```
#include <thread>
#include <vector>

// Function to handle client requests
void handle_client(int client_socket) {
    // Buffer for incoming data
    char buffer[30000] = {0};
    read(client_socket, buffer, sizeof(buffer));

    // Process the request (e.g., parse HTTP headers, serve files)
    std::string response = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n"
                           "<html><body><h1>Hello, World!</h1></body></html>";

    // Send response
    send(client_socket, response.c_str(), response.size(), 0);

    // Close the socket
    close(client_socket);
}

int main() {
    // (previous socket setup code here...)
```

```
    std::vector<std::thread> threads;

    while (true) {
        int new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen);
        if (new_socket < 0) {
            perror("accept");
            exit(EXIT_FAILURE);
        }

        // Create a new thread to handle the client
        threads.emplace_back(handle_client, new_socket);

        // Detach thread to allow it to run independently
        threads.back().detach();
    }

    return 0;
}
```

3. Serving Static HTML Files

```
#include <fstream>

void handle_client(int client_socket) {
    char buffer[30000] = {0};
    read(client_socket, buffer, sizeof(buffer));

    // Parse the HTTP request to get the requested file path
    std::string request(buffer);
```

```
std::string file_path = "index.html"; // Default file

// Check if the request contains a specific file path
auto pos = request.find("GET /");
if (pos != std::string::npos) {
    auto end_pos = request.find(' ', pos + 5);
    if (end_pos != std::string::npos) {
        file_path = request.substr(pos + 5, end_pos - pos - 5);
    }
}

// Open the requested file
std::ifstream file(file_path);
std::string response;
if (file.is_open()) {
    std::stringstream buffer;
    buffer << file.rdbuf();
    std::string file_content = buffer.str();

    response = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n" + file_content;
} else {
    response = "HTTP/1.1 404 Not Found\r\nContent-Type: text/html\r\n\r\n"
               "<html><body><h1>404 Not Found</h1></body></html>";
}

// Send response
send(client_socket, response.c_str(), response.size(), 0);

// Close the socket
```

```
    close(client_socket);
}
```
