**Task 2**

This C++ program is a Contact Management System designed to handle a list of contacts via a console-based interface. It provides users with three core functionalities: adding a new contact, removing an existing contact by its ID, and displaying all saved contacts. Contacts are stored in an array with a maximum capacity defined by `max\_entries`, which is set to 100. Each contact includes an ID, a name, and a phone number. The program offers a menu-driven interface, allowing users to choose an operation by entering the corresponding number. Users can add contacts by entering a name and phone number, delete contacts by specifying the contact ID, and view all stored contacts along with their details. The program continues to run in a loop until the user selects the exit option from the menu.

**Code:**

#include<iostream>

using namespace std;

const int max\_entries = 100;

class CONTACT

{

public:

int id;

string name;

string phonenumber;

public:

CONTACT() {

id = -1;

}

};

class contactmanagementsystem

{

private:

CONTACT contacts[max\_entries];

int contactcount;

public:

contactmanagementsystem()

{

contactcount = 0;

}

void addcontacts()

{

if (contactcount < max\_entries)

{

CONTACT contact;

contact.id = contactcount + 1;

cout << "ENTER NAME OF CONTACT:";

cin.ignore();

getline(cin, contact.name);

cout << "ENTER CONTACT PHONENUMBER:";

getline(cin, contact.phonenumber);contacts[contactcount++] = contact;

}

}

void viewcontacts()

{

cout << "CONTACTS" << endl;

for (int i = 0; i < contactcount; ++i)

{

cout << "ID OF CONTACT IS:" << contacts[i].id << endl;

cout << "NAME OF CONTACT IS:" << contacts[i].name << endl;

cout << "PHONENUMBER OF CONTACT IS:" << contacts[i].phonenumber <<

endl;

}

}

void deletecontacts() {

int id;

cout << "ENTER ID OF CONTACT TO DELETE: ";

cin >> id;

int index = -1;

for (int i = 0; i < contactcount; ++i) {

if (contacts[i].id == id) {

index = i;

}

}

--contactcount;

cout << "CONTACT WITH ID " << id << " DELETED SUCCESSFULLY." << endl;

}

};

int main()

{

contactmanagementsystem cms;

int choice = 0;

do

{

cout << "CONTACT MANAGEMENT SYSTEM" << endl;

cout << "1.ADD CONTACT" << endl;

cout << "2.DELTE CONTACT" << endl;

cout << "3.VIEW CONTACT" << endl;

cout << "enter choice" << endl;

cin >> choice;

switch (choice)

{

case 1:

cms.addcontacts();

break;

case 2:

cms.deletecontacts();

break;

case 3:

cms.viewcontacts();

case 4:

cout << "EXITING..." << endl;

break;

default:

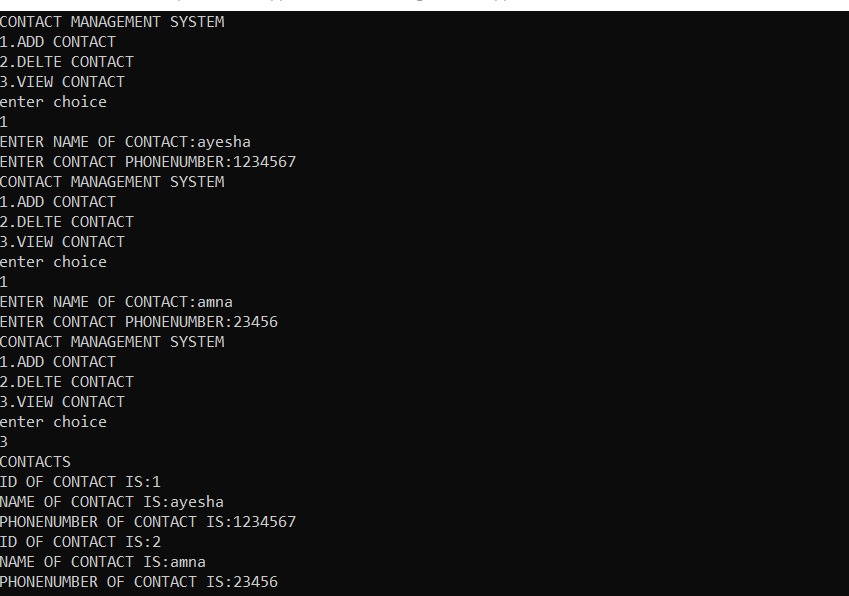
cout << "INVALID CHOICE PLZ ENTER CORRECT CHOICE" << endl;

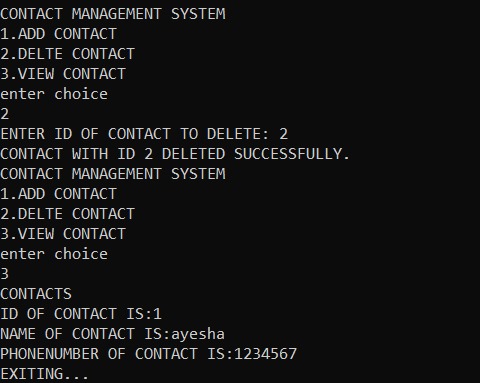
}

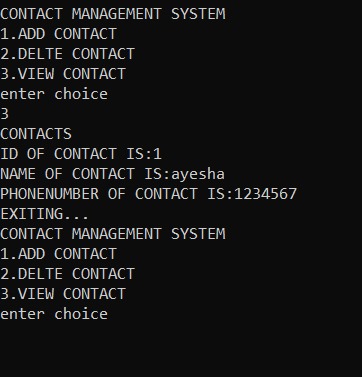
} while (choice != 4);

return 0;

}



****

****

**TASK 1**

This C++ program is a location and weather management system that allows users to manage location details, retrieve current weather, historical weather, and air quality data for specified locations. It includes classes for managing locations (`Location`), weather data (`WeatherData`), and services for fetching weather information (`WeatherService`, `HistoricalWeatherService`, `AirQualityService`). The main function demonstrates these functionalities by adding and displaying locations (Rawalpindi and Multan), updating weather data, removing a location, displaying weather information, and exporting location data to a CSV file.

**Code:**

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

#include <fstream>

using namespace std;

class Location {

private:

string name;

double longitude;

double latitude;

public:

Location() : name("unknown"), longitude(0.0), latitude(0.0) {}

Location(const string& nm, double lg, double lt) : name(nm), longitude(lg), latitude(lt) {}

void updateLocation(const string& nm, double lg, double lt) {

name = nm;

longitude = lg;

latitude = lt;

}

void displayLocation() const {

cout << "Location Name: " << name << endl;

cout << "Longitude: " << longitude << endl;

cout << "Latitude: " << latitude << endl;

}

string getName() const { return name; }

double getLongitude() const { return longitude; }

double getLatitude() const { return latitude; }

};

class WeatherData {

private:

float temperature;

float windSpeed;

public:

WeatherData() : temperature(0.0f), windSpeed(0.0f) {}

WeatherData(float temp, float wind) : temperature(temp), windSpeed(wind) {}

void setWeatherData(float temp, float wind) {

temperature = temp;

windSpeed = wind;

}

void displayWeatherData() const {

cout << "Temperature: " << temperature << " °C" << endl;

cout << "Wind Speed: " << windSpeed << " m/s" << endl;

}

};

class WeatherService {

public:

static string getWeatherInfo(const Location& loc) {

return "Weather Info for " + loc.getName() + ": 25°C, Wind Speed: 5 m/s";

}

static void showWeatherInfo(const Location& loc) {

string weatherInfo = getWeatherInfo(loc);

cout << weatherInfo << endl;

}

};

class HistoricalWeatherService {

public:

static string getHistoricalInfo(const Location& loc) {

return "Historical Weather Info for " + loc.getName() + ": 20°C, Wind Speed: 3 m/s";

}

static void showHistoricalInfo(const Location& loc) {

string historicalInfo = getHistoricalInfo(loc);

cout << historicalInfo << endl;

}

};

class AirQualityService {

public:

static string getAirQualityInfo(const Location& loc) {

return "Air Quality Info for " + loc.getName() + ": AQI 42 (Good)";

}

static void showAirQualityInfo(const Location& loc) {

string airQualityInfo = getAirQualityInfo(loc);

cout << airQualityInfo << endl;

}

};

void exportLocationsToCSV(const vector<Location>& locations, const string& filename) {

ofstream file(filename);

if (file.is\_open()) {

file << "Name,Longitude,Latitude\n";

for (const auto& loc : locations) {

file << loc.getName() << "," << loc.getLongitude() << "," << loc.getLatitude() << "\n";

}

file.close();

cout << "Data exported to " << filename << endl;

} else {

cerr << "Failed to open file " << filename << endl;

}

}

int main() {

WeatherData wd;

vector<Location> locations;

wd.setWeatherData(12.6f, 200.0f);

wd.displayWeatherData();

wd.setWeatherData(12.3f, 100.0f);

wd.displayWeatherData();

locations.emplace\_back("Rawalpindi", 73.0479, 33.6844);

locations.emplace\_back("Multan", 71.5249, 30.1575);

for (const auto& loc : locations) {

loc.displayLocation();

}

string locNameToRemove;

cout << "Enter name of location to remove: ";

cin >> locNameToRemove;

auto it = remove\_if(locations.begin(), locations.end(),

[&locNameToRemove](const Location& loc) { return loc.getName() == locNameToRemove; });

if (it != locations.end()) {

locations.erase(it, locations.end());

cout << "Location " << locNameToRemove << " successfully deleted." << endl;

} else {

cout << "Location " << locNameToRemove << " not found." << endl;

}

if (!locations.empty()) {

WeatherService::showWeatherInfo(locations.front());

HistoricalWeatherService::showHistoricalInfo(locations.front());

AirQualityService::showAirQualityInfo(locations.front());

}

exportLocationsToCSV(locations, "locations.csv");

return 0;

}

***OUTPUT***

