Assignment-1

Weather Data Storage System

Includes and Namespace

Explanation:

#include <iostream>: For input (cin) and output (cout).

#include <string>: For using string data type.

#include <vector>: For dynamic arrays (2D arrays are implemented as vectors of vectors).

using namespace std; : So, we don't have to type std:: every time.

Outcome: Allows the program to use standard input/output, strings, and vectors easily.

WeatherRecord ADT

```cpp
// Weather Record ADT
struct WeatherRecord {
    string date;
    string city;
    double temperature;

    WeatherRecord() {
        date = "";
        city = "";
        temperature = -9999; // sentinel for missing
    }

    WeatherRecord(string d, string c, double t) {
        date = d;
        city = c;
        temperature = t;
    }
};
```

Explanation:

Purpose: Stores a single weather record (ADT = Abstract Data Type).

date → Stores date of the record

city → Stores city name.

temperature → Stores temperature value.

Constructors:

Default → Initializes empty date/city and -9999 for temperature (indicates no data).

Parameterized → Initializes a record with given date, city, and temperature.

Outcome: You can create objects like WeatherRecord w; or WeatherRecord w("01/01/2023", "Delhi", 25.5);.

Functionality: Represents each individual weather data entry.

WeatherDataSystem Class

```
// Weather Data System
class WeatherDataSystem {
private:
    int numYears, numCities;
    vector<string> years;
    vector<string> cityNames;
    vector<vector<WeatherRecord>> records; // 2D array
```

Explanation:

numYears, numCities → Stores number of years and cities.

years → Vector storing all year strings.

cityNames → Vector storing all city names.

records → 2D array (vector of vectors) storing WeatherRecord objects for each year-city combination.

Outcome: Sets up the structure to store and manage multiple weather records.

Constructor

```
public:
    WeatherDataSystem() {
        numYears = 0;
        numCities = 0;
    }
```

Explanation: Initializes years and cities to 0.

Outcome: Ensures safe default initialization before user inputs.

Setup Function

```
void setup() {
    cout << "Enter number of years: ";
    cin >> numYears;
    cout << "Enter number of cities: ";
    cin >> numCities;

    cin.ignore();
    years.resize(numYears);
    cityNames.resize(numCities);
    records.resize(numYears, vector<WeatherRecord>(numCities));

    cout << "Enter years:\n";
    for(int i = 0; i < numYears; i++) cin >> years[i];

    cin.ignore();
    cout << "Enter city names:\n";
    for(int i = 0; i < numCities; i++) getline(cin, cityNames[i]);
}

int getYearIndex(string year) {
    for(int i = 0; i < numYears; i++)
        if(years[i] == year) return i;
    return -1;
}
```

Explanation:

Takes input from the user: number of years and cities.

Resizes vectors to hold records.

Inputs years and city names.

Initializes the records 2D array.

Outcome: Prepares the system to store weather data.

Functionality: Initializes the grid (2D array) to hold weather records.

Helper Functions

```
int getYearIndex(string year) {
    for(int i = 0; i < numYears; i++)
        if(years[i] == year) return i;
    return -1;
}

int getCityIndex(string city) {
    for(int i = 0; i < numCities; i++)
        if(cityNames[i] == city) return i;
    return -1;
}
```

Explanation:

Maps a year or city name to the corresponding row/column index in the 2D array.

Returns -1 if the year/city is invalid.

Outcome: Lets program know where to insert, delete, or retrieve records in the 2D array.

ADT Operations

Insert Weather Record

```
// --- ADT Methods ---
void insertWeatherRecord() {
    string date, city;
    double temp;
    cout << "Enter date (DD/MM/YYYY): ";
    cin >> date;
    cin.ignore();
    cout << "Enter city: ";
    getline(cin, city);
    cout << "Enter temperature: ";
    cin >> temp;

    int row = getYearIndex(date.substr(6,4));
    int col = getCityIndex(city);

    if(row >= 0 && col >= 0) {
        records[row][col] = WeatherRecord(date, city, temp);
        cout << "Record inserted successfully!\n";
    } else {
        cout << "Invalid year or city!\n";
    }
}
```

Explanation:

Inputs date, city, temperature.

Finds row = year index, col = city index.

Inserts record in 2D array if valid.

Outcome: Successfully adds a weather record.

Functionality: Uses ADT object to store structured data.

Delete Weather Record

```
void deleteWeatherRecord() {
    string date, city;
    cout << "Enter date (DD/MM/YYYY) to delete: ";
    cin >> date;
    cin.ignore();
    cout << "Enter city: ";
    getline(cin, city);

    int row = getYearIndex(date.substr(6,4));
    int col = getCityIndex(city);

    if(row >= 0 && col >= 0 && records[row][col].temperature != -9999) {
        records[row][col] = WeatherRecord();
        cout << "Record deleted successfully!\n";
    } else {
        cout << "Record not found!\n";
    }
}
```

Explanation:

Inputs date, city, temperature.

Finds row = year index, col = city index.

Inserts record in 2D array if valid.

Outcome: Successfully adds a weather record.

Functionality: Uses ADT object to store structured data.

Retrieve Weather Record

```
void retrieveWeatherRecord() {
    string city, year;
    cin.ignore();
    cout << "Enter city: ";
    getline(cin, city);
    cout << "Enter year: ";
    cin >> year;

    int row = getYearIndex(year);
    int col = getCityIndex(city);

    if(row >= 0 && col >= 0 && records[row][col].temperature != -9999) {
        cout << "Date: " << records[row][col].date
             << ", City: " << records[row][col].city
             << ", Temperature: " << records[row][col].temperature << "°C\n";
    } else {
        cout << "No record found.\n";
    }
}
```

Explanation:

Inputs date, city, temperature.

Finds row = year index, col = city index.

Inserts record in 2D array if valid.

Outcome: Successfully adds a weather record.

Functionality: Uses ADT object to store structured data.

2D Array Specific Functions

Insert Temperature

```
// --- 2D Array Methods ---
void populateArrary() {
    string year, city, date;
    double temp;
    cin.ignore();
    cout << "Enter year: ";
    getline(cin, year);
    cout << "Enter city: ";
    getline(cin, city);
    cout << "Enter date (DD/MM/YYYY): ";
    getline(cin, date);
    cout << "Enter temperature: ";
    cin >> temp;

    int row = getYearIndex(year);
    int col = getCityIndex(city);

    if(row >= 0 && col >= 0) {
        records[row][col] = WeatherRecord(date, city, temp);
        cout << "Temperature inserted successfully!\n";
    } else {
        cout << "Invalid year or city!\n";
    }
}
```

Explanation: Similar to insertWeatherRecord(), but focuses on year + city, not full date.

Outcome: Demonstrates 2D array manipulation directly.

Row-Major Access

```
void rowMajorAccess() {
    cout << "\nRow-Major Access:\n";
    for(int i = 0; i < numYears; i++)
        for(int j = 0; j < numCities; j++)
            if(records[i][j].temperature != -9999)
                cout << records[i][j].date << " - "
                     << records[i][j].city << " : "
                     << records[i][j].temperature << "°C\n";
}
```

Explanation: Traverses 2D array row by row (year by year).

Outcome: Displays all records in row-major order.

Column-Major Access

```
void columnMajorAccess() {
    cout << "\nColumn-Major Access:\n";
    for(int j = 0; j < numCities; j++)
        for(int i = 0; i < numYears; i++)
            if(records[i][j].temperature != -9999)
                cout << records[i][j].date << " - "
                     << records[i][j].city << " : "
                     << records[i][j].temperature << "°C\n";
}
```

Explanation: Traverses column by column (city by city).

Outcome: Shows how data can be accessed differently in 2D arrays.

Handle Sparse Data

```
void handleSparseData() {
    cout << "\nSparse Data (Missing Records):\n";
    for(int i = 0; i < numYears; i++)
        for(int j = 0; j < numCities; j++)
            if(records[i][j].temperature == -9999)
                cout << "Year: " << years[i] << ", City: " << cityNames[j] << " -> Missing Data\n";
}
```

Explanation: Traverses column by column (city by city).

Outcome: Shows how data can be accessed differently in 2D arrays.

Complexity Analysis

```
    void complexityAnalysis() {
    cout << "\n--- Complexity Analysis ---\n";

    // Theoretical complexities
    cout << "Time Complexity (Theoretical):\n";
    cout << " Insert/Delete/Retrieve: O(1)\n";
    cout << " Row/Column Traversal: O(years * cities)\n";

    cout << "Space Complexity (Theoretical):\n";
    cout << " Array Storage: O(years * cities)\n";

    // Actual values
    cout << "\nTime Complexity (Actual Values):\n";
    cout << " Insert/Delete/Retrieve: Constant time = 1 step\n";
    cout << " Row/Column Traversal: O(" << numYears << " * " << numCities
         << ") = " << numYears * numCities << " steps\n";

    cout << "\nSpace Complexity (Actual Values):\n";
    cout << " Array size = " << numYears << " * " << numCities
         << " = " << numYears * numCities << " cells\n";
}
};
```

Explanation:

Shows theoretical and actual time/space complexity.

O(1) for single record operations, O(years * cities) for traversals.

Outcome: Helps understand performance characteristics of the system.

Main Program

```cpp
// --- Main Program ---
int main() {
    WeatherDataSystem system;
    system.setup();

    int choice;
    do {
        cout << "\n--- Weather Data System Menu ---\n";
        cout << "1. Insert Weather Record (ADT)\n";
        cout << "2. Delete Weather Record (ADT)\n";
        cout << "3. Retrieve Weather Record (ADT)\n";
        cout << "4. Insert Temperature (2D Array)\n";
        cout << "5. Row Major Access\n";
        cout << "6. Column Major Access\n";
        cout << "7. Handle Sparse Data\n";
        cout << "8. Complexity Analysis\n";
        cout << "0. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;

        switch(choice) {
            case 1: system.insertWeatherRecord(); break;
            case 2: system.deleteWeatherRecord(); break;
            case 3: system.retrieveWeatherRecord(); break;
            case 4: system.populateArrary(); break;
            case 5: system.rowMajorAccess(); break;
            case 6: system.columnMajorAccess(); break;
            case 7: system.handleSparseData(); break;
            case 8: system.complexityAnalysis(); break;
            case 0: cout << "Exiting program...\n"; break;
            default: cout << "Invalid choice!\n";
        }
    } while(choice != 0);

    return 0;
}
```

Explanation:

Menu-driven program.

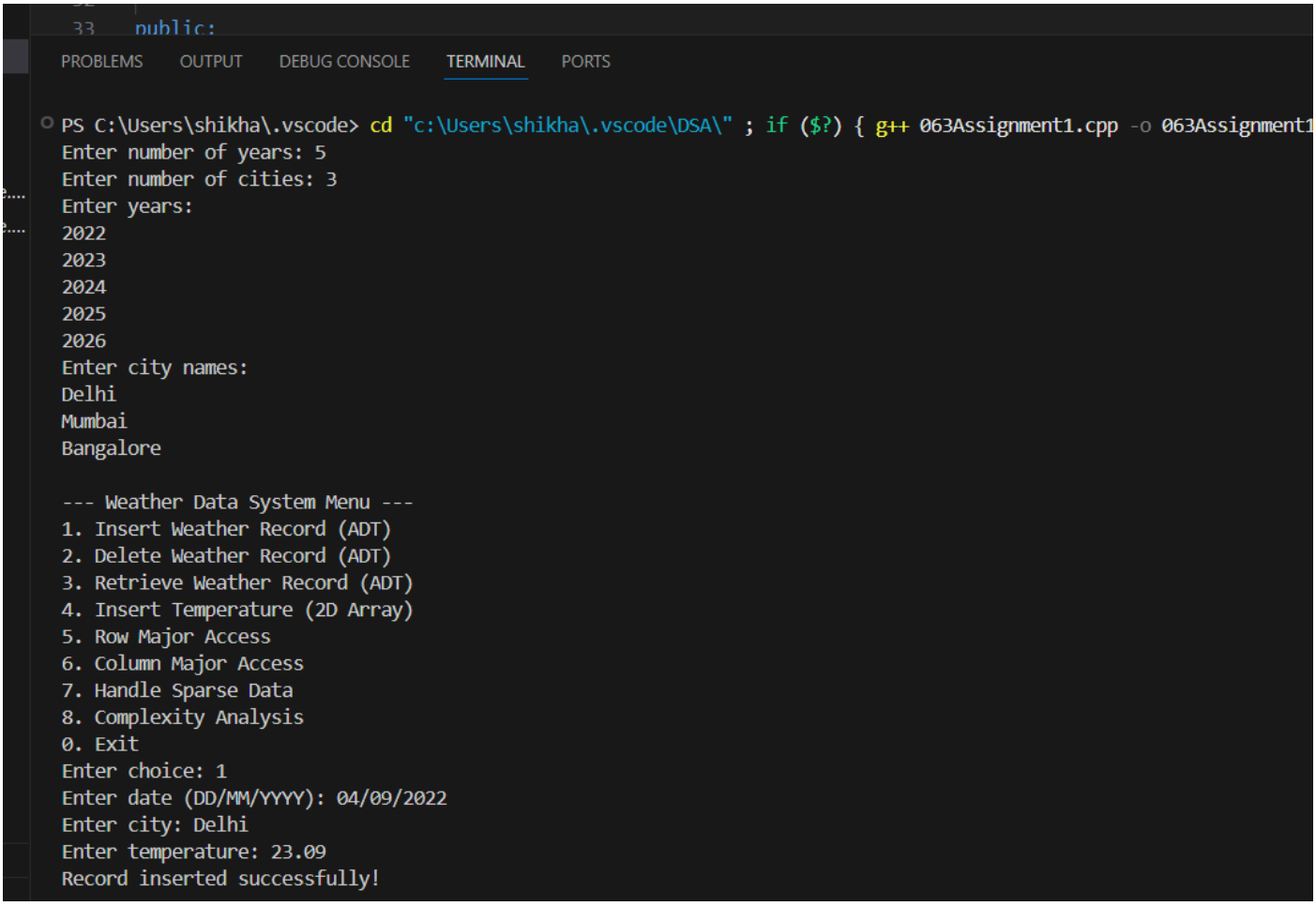
Uses do-while loop to continuously take user input until 0 (exit).

Each menu option calls the corresponding class function.

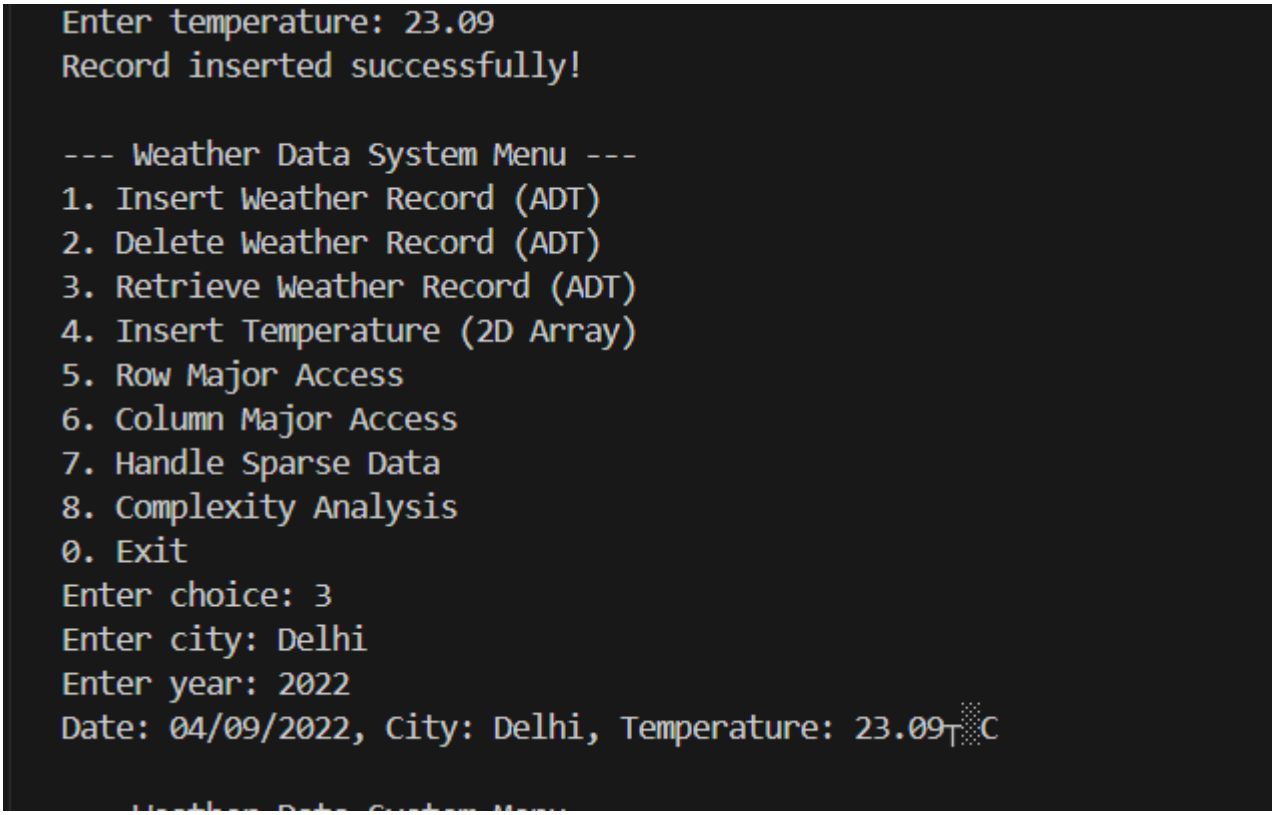
Outcome: User can insert, delete, retrieve, traverse, and analyse weather data interactively.

Test Cases

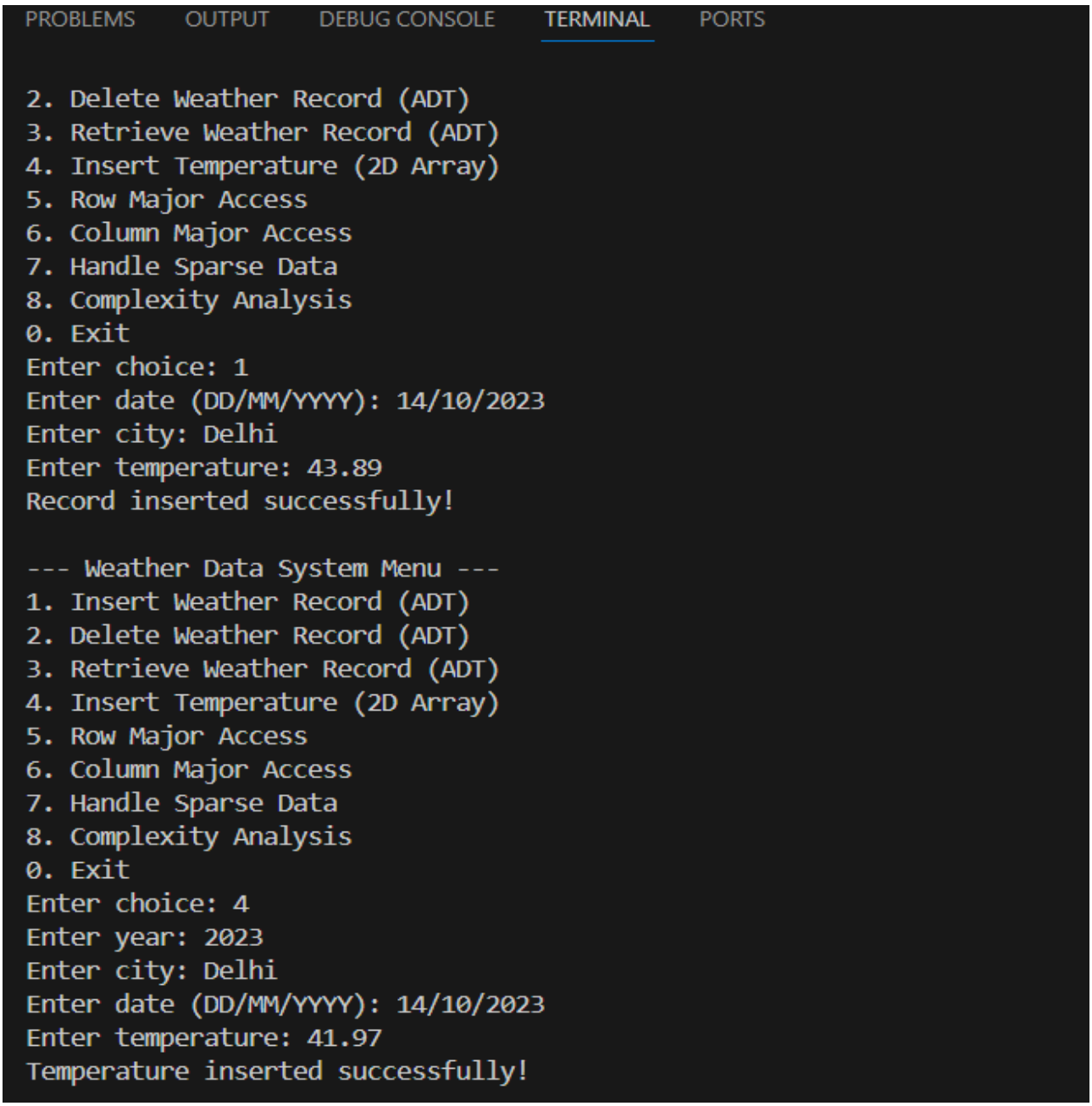
Test Case: 1 Insert Weather Record



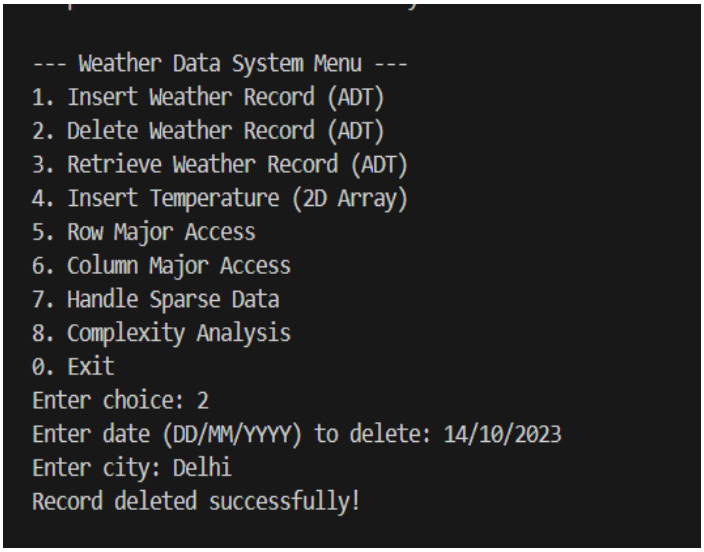
Test Case: 2 Retrieve Weather Record (ADT)



Test Case: 3 Insert Temperature (2D Array)



Test Case: 4 Delete Weather Record



Test Case: 5 Row Major Access

```
--- Weather Data System Menu ---
1. Insert Weather Record (ADT)
2. Delete Weather Record (ADT)
3. Retrieve Weather Record (ADT)
4. Insert Temperature (2D Array)
5. Row Major Access
6. Column Major Access
7. Handle Sparse Data
8. Complexity Analysis
0. Exit
Enter choice: 5

Row-Major Access:
04/09/2022 - Delhi : 23.09┬ C
12/12/2023 - Mumbai : 32.9┬ C
```

Test Case: 6 Column Major Access

```
 --- Weather Data System Menu ---
 1. Insert Weather Record (ADT)
 2. Delete Weather Record (ADT)
 3. Retrieve Weather Record (ADT)
 4. Insert Temperature (2D Array)
 5. Row Major Access
 6. Column Major Access
 7. Handle Sparse Data
 8. Complexity Analysis
 0. Exit
 Enter choice: 6

 Column-Major Access:
 04/09/2022 - Delhi : 23.09┬ C
 12/12/2023 - Mumbai : 32.9┬ C
```

Test Case: 7 Handle Sparse Data

```
--- Weather Data System Menu ---
1. Insert Weather Record (ADT)
2. Delete Weather Record (ADT)
3. Retrieve Weather Record (ADT)
4. Insert Temperature (2D Array)
5. Row Major Access
6. Column Major Access
7. Handle Sparse Data
8. Complexity Analysis
0. Exit
Enter choice: 7

Sparse Data (Missing Records):
Year: 2022, City: Mumbai -> Missing Data
Year: 2022, City: Bangalore -> Missing Data
Year: 2023, City: Delhi -> Missing Data
Year: 2023, City: Bangalore -> Missing Data
Year: 2024, City: Delhi -> Missing Data
Year: 2024, City: Mumbai -> Missing Data
Year: 2024, City: Bangalore -> Missing Data
Year: 2025, City: Delhi -> Missing Data
Year: 2025, City: Mumbai -> Missing Data
Year: 2025, City: Bangalore -> Missing Data
Year: 2026, City: Delhi -> Missing Data
Year: 2026, City: Mumbai -> Missing Data
Year: 2026, City: Bangalore -> Missing Data
```

Test Case: 8 Complexity Analysis

```
--- Weather Data System Menu ---
1. Insert Weather Record (ADT)
2. Delete Weather Record (ADT)
3. Retrieve Weather Record (ADT)
4. Insert Temperature (2D Array)
5. Row Major Access
6. Column Major Access
7. Handle Sparse Data
8. Complexity Analysis
0. Exit
Enter choice: 8

--- Complexity Analysis ---
Time Complexity (Theoretical):
 Insert/Delete/Retrieve: O(1)
 Row/Column Traversal: O(years * cities)
Space Complexity (Theoretical):
 Array Storage: O(years * cities)

Time Complexity (Actual Values):
 Insert/Delete/Retrieve: Constant time = 1 step
 Row/Column Traversal: O(5 * 3) = 15 steps

Space Complexity (Actual Values):
 Array size = 5 * 3 = 15 cells
```