

*Course : Data Structures*

*Semester: 3rd*

*Session: 2025-26*

*Submitted By: Japneet Kaur Roll No:2401720015*

*Course: B.Sc.(Hons)computer science*

*Submitted to :Deepak Kaushik*

**Problem Statement**

The problem is to develop a **Weather Data Storage System** that efficiently collects, stores, and manages weather-related information—specifically temperature records—organized by **date** and **city**. The system should utilize **structured data formats** such as **2D arrays** and **Abstract Data Types (ADTs)** to support systematic data handling. It must provide functionalities for **inserting**, **retrieving**, and **analyzing** weather data, ensuring efficient access to both **complete** and **sparse datasets**. The solution should enable users to observe trends, compare data across cities and dates, and maintain an organized repository of weather information for long-term analysis.

**Objectives**

* To design a structured system for storing weather data (temperature) by **city** and **date**.
* To implement **2D arrays** and **ADTs** for efficient data organization.
* To enable easy **insertion**, **retrieval**, and **updating** of weather records.
* To support analysis of weather trends across different cities and time periods.
* To handle both **complete** and **sparse** datasets efficiently.

**WeatherRecord (ADT)**

**Attributes:**

* **date**: Stores the date of the weather record.
* **city**: Holds the name of the city.
* **temperature**: Contains the temperature value for the given date and city.

**Methods:**

* **WeatherRecord()**: Initializes attributes with default values.
* **WeatherRecord(string d, string c, double t)**: Creates a record with given date, city, and temperature.

**WeatherDataStorage (Class)**

**Attributes:**

* **cities**: List of all cities being recorded.
* **years**: List of all years considered for data storage.
* **temperatureData**: 2D array storing temperature values by year and city.
* **sentinel**: Special marker value for missing data.

**Methods:**

* **WeatherDataStorage(vector<int>, vector<string>)**: Initializes the data storage system.
* **insertRecord(WeatherRecord)**: Inserts a weather record into the system.
* **deleteRecord(string, int)**: Removes a record by marking it with a sentinel value.
* **retrieveRecord(string, int)**: Fetches temperature data for a given city and year.
* **populateArray()**: Fills the array with sample data.
* **rowMajorAccess()**: Displays data in year-wise order.
* **columnMajorAccess()**: Displays data in city-wise order.
* **handleSparseData()**: Identifies missing records in the dataset.
* **analyzeComplexity()**: Displays time and space complexity of operations.
* **getYearIndex(int)**: Finds the index of a specific year.
* **getCityIndex(string)**: Finds the index of a specific city.

|  |  |
| --- | --- |
| Row-major | Accessing year-wise(row-by-row) |
| Column-major | Accessing city-wise(column-by-colum) |

| **Operation / Component** | **Time Complexity** | **Space Complexity** | **Notes** |
| --- | --- | --- | --- |
| Insert Record | O(1) | — | Direct index access in 2D array |
| Delete Record | O(1) | — | Mark with sentinel value |
| Retrieve Record | O(1) | — | Access by year and city indices |
| Row-Major / Column-Major Access | O(n \* m) | — | Traverse entire 2D array |
| Handle Sparse Data | O(n \* m) | — | Check for sentinel values |
| 2D Array (temperatureData) | — | O(n \* m) | n = years, m = cities |
| Cities List | — | O(m) | Store city names |
| Years List | — | O(n) | Store years |

Code:

#include <iostream>

#include <vector>

#include <string>

#include <iomanip>

using namespace std;

// -------------------- Weather Record ADT --------------------

class WeatherRecord {

public:

string date;

string city;

double temperature;

WeatherRecord() : date(""), city(""), temperature(0.0) {}

WeatherRecord(string d, string c, double t) : date(d), city(c), temperature(t) {}

};

// -------------------- Data Storage Class --------------------

class WeatherDataStorage {

private:

vector<string> cities;

vector<int> years;

vector<vector<double>> temperatureData; // 2D array: rows = years, cols = cities

double sentinel = -9999.0; // Sentinel for sparse data

public:

// Constructor

WeatherDataStorage(vector<int> y, vector<string> c) {

years = y;

cities = c;

temperatureData.resize(years.size(), vector<double>(cities.size(), sentinel));

}

// Insert new weather record

void insertRecord(WeatherRecord record) {

int year = stoi(record.date.substr(6, 4)); // extract year from dd-mm-yyyy

int rowIndex = getYearIndex(year);

int colIndex = getCityIndex(record.city);

if (rowIndex != -1 && colIndex != -1) {

temperatureData[rowIndex][colIndex] = record.temperature;

cout << "Inserted: " << record.city << " (" << record.date << ") = "

<< record.temperature << "°C\n";

} else {

cout << "Invalid city or year.\n";

}

}

// Delete record

void deleteRecord(string city, int year) {

int rowIndex = getYearIndex(year);

int colIndex = getCityIndex(city);

if (rowIndex != -1 && colIndex != -1) {

temperatureData[rowIndex][colIndex] = sentinel;

cout << "Deleted record for " << city << " in " << year << ".\n";

} else {

cout << "Record not found.\n";

}

}

// Retrieve data for a city in a specific year

void retrieveRecord(string city, int year) {

int rowIndex = getYearIndex(year);

int colIndex = getCityIndex(city);

if (rowIndex != -1 && colIndex != -1) {

double temp = temperatureData[rowIndex][colIndex];

if (temp != sentinel)

cout << "Temperature for " << city << " in " << year << ": " << temp << "°C\n";

else

cout << "No data available for " << city << " in " << year << ".\n";

} else {

cout << "Invalid city or year.\n";

}

}

// Populate the array with data (for demo)

void populateArray() {

cout << "\nPopulating array with sample data...\n";

insertRecord(WeatherRecord("01-01-2024", "Delhi", 25.3));

insertRecord(WeatherRecord("01-01-2024", "Mumbai", 30.1));

insertRecord(WeatherRecord("01-01-2025", "Delhi", 24.5));

insertRecord(WeatherRecord("01-01-2025", "Mumbai", 29.9));

}

// Row-major access

void rowMajorAccess() {

cout << "\nRow-Major Access (Year-wise):\n";

for (size\_t i = 0; i < years.size(); ++i) {

cout << "Year " << years[i] << ": ";

for (size\_t j = 0; j < cities.size(); ++j) {

cout << setw(8);

if (temperatureData[i][j] != sentinel)

cout << temperatureData[i][j];

else

cout << "N/A";

}

cout << "\n";

}

}

// Column-major access

void columnMajorAccess() {

cout << "\nColumn-Major Access (City-wise):\n";

for (size\_t j = 0; j < cities.size(); ++j) {

cout << "City " << cities[j] << ": ";

for (size\_t i = 0; i < years.size(); ++i) {

cout << setw(8);

if (temperatureData[i][j] != sentinel)

cout << temperatureData[i][j];

else

cout << "N/A";

}

cout << "\n";

}

}

// Handle sparse data

void handleSparseData() {

cout << "\nSparse Data Handling:\n";

for (size\_t i = 0; i < years.size(); ++i) {

for (size\_t j = 0; j < cities.size(); ++j) {

if (temperatureData[i][j] == sentinel)

cout << "Missing data for " << cities[j] << " in " << years[i] << ".\n";

}

}

}

// Analyze complexity (theoretical)

void analyzeComplexity() {

cout << "\nComplexity Analysis:\n";

cout << "Insert: O(1) [direct index access]\n";

cout << "Delete: O(1)\n";

cout << "Retrieve: O(1)\n";

cout << "Space: O(n \* m) where n = years, m = cities\n";

}

private:

int getYearIndex(int year) {

for (size\_t i = 0; i < years.size(); ++i)

if (years[i] == year)

return i;

return -1;

}

int getCityIndex(string city) {

for (size\_t i = 0; i < cities.size(); ++i)

if (cities[i] == city)

return i;

return -1;

}

};

// -------------------- Main Function --------------------

int main() {

vector<int> years = {2024, 2025};

vector<string> cities = {"Delhi", "Mumbai"};

WeatherDataStorage storage(years, cities);

storage.populateArray();

storage.rowMajorAccess();

storage.columnMajorAccess();

storage.retrieveRecord("Delhi", 2024);

storage.deleteRecord("Delhi", 2024);

storage.retrieveRecord("Delhi", 2024);

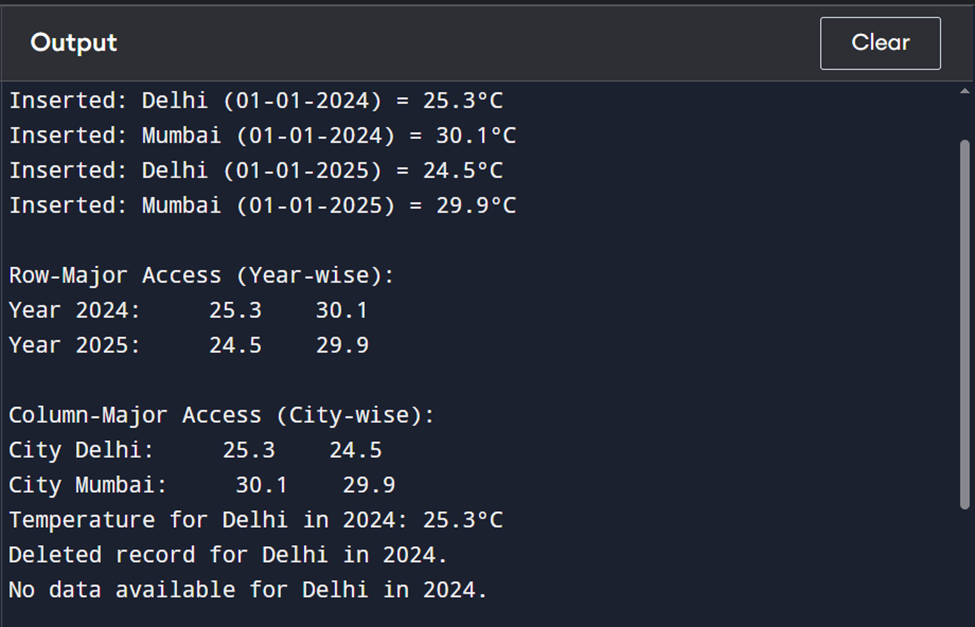
storage.handleSparseData();

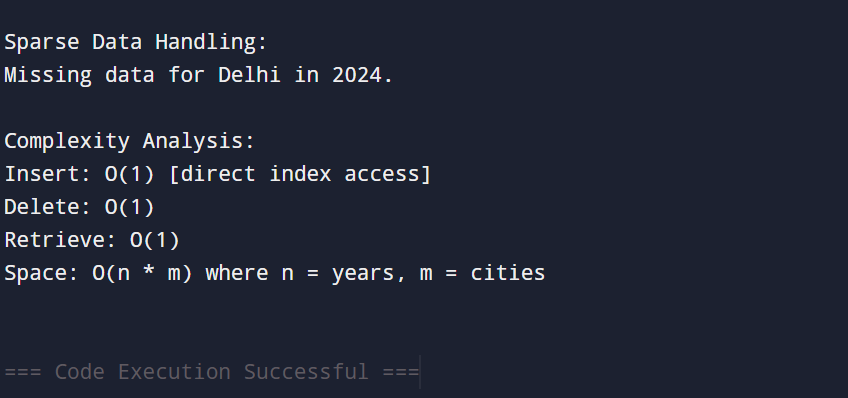
storage.analyzeComplexity();

return 0;

}

Output:





**Conclusion (Key Points)**

* The system efficiently **stores and manages temperature data** for multiple cities over multiple years.
* **2D arrays** and **Abstract Data Types (ADTs)** are used for structured and organized data storage.
* **Row-major and column-major access** allows flexible data analysis:
  + Row-major → year-wise comparisons
  + Column-major → city-wise trends
* **Insertion, deletion, and retrieval operations** are performed in **O(1) time**, ensuring fast access.
* **Sparse datasets** are handled effectively using **sentinel values** to mark missing data.
* The system demonstrates **memory-efficient storage** and clear **time and space complexity analysis**.
* Provides practical experience in **applying data structures** to real-world scenarios and strengthens programming skills.