NATIONAL UNIVERSITY OF COMPUTER & EMERGING SCIENCES ISLAMABAD

Due Date ≈ Friday, March 26th, 2025 (11:59 pm)

Instructions

Submission: Combine all your work in a **.zip** file, it should only contain code (.cpp files) or data (.txt files). Use proper naming convention for your submission file SECTION_ROLL-NUM_01.zip (e.g. P_24i0412_01.zip). Failure to submit according to the above format would result in 25% marks deduction. Only on Google Classroom!

Plagiarism: Plagiarism cases will be dealt with strictly. If found plagiarized, both the involved parties will be awarded zero marks in this assignment, all the remaining assignments, or even an F grade in the course.

Deadline: The deadline is mentioned above. Late submission with marks deduction will be accepted according to the course policy shared earlier.

Bonus: Task 5 is the bonus task.

Note:

- Additional marks may be awarded for using good programming practices, including memory efficient programs, well-written, good design and properly commented.
- All programs must be generic.
- Follow the given instructions to the letter, failing to do so will result in a zero.

Technical Guidelines

Struct-Centered Design:

- The core logic must revolve around structs that model stores, sales records, clustering results, rankings or more.
- Each struct should be efficiently designed to support operations such as sorting, filtering, and aggregations.
- Encapsulation and modularity are required—use functions for each major operation.

Persistent Data Storage:

- Implement file based I/O for storing and retrieving store data across executions.
- Allow the system to load previous sales history when restarted.

Demonstration in main (): Your main () function must execute a full simulation, demonstrating:

1. Loading or generating structured store data (at least 100 stores).
2. Applying geographic clustering and generating cluster reports.
3. Computing composite performance scores and ranking stores dynamically.
4. Predicting next month's sales.
5. Visualizing key sales trends using ASCII-based graphs.

UrbanEase: Store Analytics and Advanced Management System

Introduction

UrbanEase is a fast-growing fashion retail brand, rapidly expanding its footprint across Pakistan's major cities, including Karachi, Lahore, Islamabad, and Faisalabad. With its trendy offerings and a strong customer base of young professionals, UrbanEase has seen a surge in both physical and online sales.

However, as the business scales, managing store performance, optimizing logistics, and understanding customer behavior has become increasingly complex. The leadership team requires real-time insights to identify top-performing outlets, forecast sales trends, and streamline inventory distribution. To stay ahead in a competitive market, UrbanEase needs an advanced, structured analytics system—one that not only tracks store data efficiently but also extracts meaningful patterns, identifies regional opportunities, and enables data-driven decision-making.

Problem Overview

Your task is to design and implement a high-performance C++ application that extracts valuable insights from structured store data. From the start, you will need to think about how to structure your data effectively, ensuring that all operations—from analysis to visualization—are built on a foundation of well-defined, efficient representations.

This assignment **requires all data representations to be implemented using structs**, ensuring a structured and scalable approach to handling information. A well-thought-out struct-based design will enable efficient data organization, retrieval, and manipulation, forming the backbone of the entire system.

Task 1 - Advanced Structured Data Representation

UrbanEase operates hundreds of outlets, each with its own sales history, location, and management details. The system must efficiently store and process this information using well-structured data models.

Each store has:

- Basic details: Name, unique ID, and geographic coordinates (latitude, longitude).
- Sales data: Monthly sales records for the past two years (24 months).
- Manager details: Each store is managed by a person responsible for its operations.
- Operating costs: Monthly operational expenses that affect profitability calculations.

Constraints & Challenges:

- Your data structures must be efficient and extensible, capable of handling dynamic data updates (e.g., adding future sales records).
- Implement file-based storage and retrieval to allow data persistence beyond a single program execution.

```
struct Store{
   char* sName;
   int sID;
   GeoCoordinates Location;
   double sMonthlyCost;
   Employee Manager;
   Employee* Staff;
   StoreAnalytics sA;//definition given with task 3
   SalesForecast sF;//definition given with task 4
   int customerCountMonthly[24];
   ...
};
struct GeoCoordinates{
   double gLat;
   double gLong;
   ...
};
```

```
struct Employees{
   int eID;
   char* eName;
   double salary;
   ...
};

struct Products
{
   char* SKU;
   char* pName;
   double unitPrice;
};

struct SalesData
{
   Store ID;
   Products*;
   string Date
   string Time;
   double TotalSaleAmount;
   ...
}
```

Task 2 - Multi-Layered Geographic Clustering for Logistics Optimization

To manage its growing network of stores more effectively, UrbanEase needs to group stores intelligently. By clustering stores, the company can optimize inventory distribution, plan regional promotions, and improve logistics efficiency.

UrbanEase wants to enhance its regional operations by clustering stores based on two key factors:

- Geographic Proximity – Stores that are physically close together should be grouped to optimize inventory shipments, marketing efforts, and regional management.
- Performance Tiers – Within each geographic cluster, stores should be further grouped based on sales revenue and profitability to identify top-performing, average, and struggling stores for more targeted strategies.

Clustering Basics

Clustering is a technique where you group things based on certain similarity. In this assignment, we want to group stores based on location and, secondly, based on their performance. There are various known clustering algorithms, which you will learn later. For now, use the following steps to cluster stores based on the criteria mentioned above. *For the ones who are curious, this is kMeans clustering algorithm, where we try to group things around a center point (centroid).*

Pseudocode:

```
1. Euclidean distance
        $$\mathrm{d}(X_i, X_j) = \sqrt{\sum_{f=1}^{F} (x_{fi} - x_{fj})^2}$$
   where F is the number of features
2. Randomly initialize cluster centers, depending on how many clusters you want!
3. Assigning stores to clusters
        $C_i = \arg\min_j d(X_i, C_j)$
4. Centroid Update
        $C_i^{new} = \frac{1}{N}\sum_{j=1}^{N} X_j$
5. Repeat (1 to 4) until centroid converge
    a. Recalculate Euclidean distances and reassign points to clusters.
    b. Recompute centroids.
    c. Stop when centroids do not change, or changes are below a threshold.
```

Example:

Let's consider following store sales data and see how the above pseudocode work!

Store	Total Sales	Avg. Sales/Day	Number of Customers
1	500,000	2,500	10,000
2	700,000	3,500	15,000
3	300,000	1,500	7,000
4	450,000	2,200	9,500
5	600,000	3,100	13,000

Each store is represented as a 3D vector:

$X_i = (x_1, x_2, x_3)$

Where:

- x_1 = Total Sales
- x_2 = Avg. Sales/Day
- x_3 = Number of Customer

Following the pseudocode above:

1. **Compute euclidean distance between stores**

   $\mathbf{d}(X_i, X_j) = \sqrt{(x_{1i} - x_{1j})^2 + (x_{2i} - x_{2j})^2 + (x_{3i} - x_{3j})^2}$

   Example calculation

Find the distance between Store 1 (500000,2500,10000) and Store 2 (700000,3500,15000):

d(*1,2*) = $\sqrt{(500000 - 700000)^2 + (2500 - 3500)^2 + (10000 - 15000)^2}$

= $\sqrt{(-200000)^2 + (-1000)^2 + (-5000)^2}$

= **200065**

2. **Select k initial centroids randomly from the dataset**

   Example centroids from the data above for k = 3 (number of clusters)

   - C_1 = (500000,2500,10000)
   - C_2 = (700000,3500,15000)
   - C_3 = (300000,1500,7000)

3. **Assign each store to the nearest centroid/cluster**

   For each store X_i, compute the Euclidean distance to each centroid and assign it to the closest one, using the same equation give in step 1.

   For store 1 you will calculate d(1, C_1), d(1, C_2), D(1, C_3)

   Assign store to the cluster with the least distance.

4. **Centroid update**

   For each cluster, compute the new centroid based on stores assigned to each cluster using

   $$C_i^{new} = \frac{1}{N}\sum_{j=1}^{N} X_j$$

   If store 1 and 4 belong to Cluster 1:

   $$C_1^{new} = \frac{(500000{,}2500{,}10000) + (450000{,}2200{,}9500)}{2}$$

   = $(\frac{500000 + 450000}{2}, \frac{2500 + 2200}{2}, \frac{10000 + 9500}{2})$

   = (475000, 2350, 9750)

   This is the new centroid which will be used in the next iteration

Repeat (1 to 4) until centroid converge

Constraints & Challenges:

- Determine an effective way to group stores geographically—whether using a fixed radius, nearest neighbors, or a more adaptive approach.
- Implement two-tier clustering where stores are first grouped by distance and then sub-clustered based on profitability.
- Provide a summary report for each cluster, detailing store count, and total revenue.
- Ensure clustering is dynamic, allowing new stores to be added without recomputing everything from scratch.

```
struct Cluster{
  char* cID;
  char* Name;
  subClusters* subClusterList;
  Centroid center;
```

```
struct subCluster{
  char* ID;
  char* Name;
  Stores* storesList;
  Centroid subCenter;
```

```
   …                                                   …
};                                                  };

struct Centroid
   //define centroid looking at your data/features
   you want to use for clustering

   …
};
```

Task 3 - Performance Analysis & Ranking with Composite Metrics

Store rankings should be based on multiple performance factors, not just sales volume. UrbanEase considers:

- Total annual sales
- Average monthly revenue growth over the last year
- Profitability (Revenue vs. Operating Costs)

Each factor contributes differently to the overall ranking. For instance, profitability should weigh more than total sales.

Constraints & Challenges:

- Develop a custom ranking algorithm that uses weighted scoring for performance factors.
- Ensure that rankings are sortable by different criteria (e.g., sort by profit, then by growth rate).
- Generate a performance report highlighting:
    - Top 10 stores based on overall ranking.
    - Bottom 5 stores (to identify struggling locations).

```
struct StoreAnalytics{                              struct Report{
   double totalsales;                                  store* topStores;
   double totaloperationalcost;                        store* bottomStores;
   double profit;                                   };

   …
};
```

Task 4 - Predictive Sales Forecasting with Trend Detection

UrbanEase wants to predict next month's sales for each store based on past data. This will help managers make better decisions about stock, promotions, and store improvements. Instead of using basic averages you can use moving averages to predict sales for a coming month. Your system should:

- Look at past sales trends to estimate future performance.
- Detect seasonal patterns, like sales increases during Eid or holiday seasons.
- Adjust predictions for recent changes—if a store's sales are increasing or decreasing, the prediction should reflect that.

Constraints & Challenges:

- Find a way to predict next month's sales based on existing data—there's no fixed formula, but your approach should make sense and be based on past trends.
- Consider seasonality and recent changes—if a store's sales have been rising, expect them to continue rising (or vice versa).
- Generate a simple forecast report that includes:
  - Expected sales for the next month.
  - A confidence level (0-100%) to show how reliable the prediction is.
  - A warning flag for stores that may see a drop in sales

```
struct Forecast{
  char* time/month;
  double predictedSales,;
  ...
};
```

Task 5 - Bonus - Console-Based Graphical Sales Visualization

Besides numerical reports, UrbanEase requires visual insights in the console. Using ASCII-based charts, the system should display:

- Monthly sales trends for each store.
- Comparative store performance (e.g., bar graphs showing different stores' profits side by side).
- Cluster-wise profit distribution (e.g., which cluster contributes the most to total revenue).

New Constraints & Challenges:

- Implement dynamic scaling for console-based bar charts (ensure visibility across different screen sizes).
- Allow users to select a store for visualization rather than printing everything at once.