
PRACTICAL ASSIGNMENT

ARRAYS & COLLECTIONS IN C#

PROBLEM STATEMENT

Enterprise Data Processing & Control System

An enterprise system processes **products, branch sales, customer transactions, financial records, task flows, and legacy data**.
You are required to simulate this system using **C# arrays and collections**, handling **dynamic input**, **data validation**, and **cross-dependency between data structures**.

GENERAL INSTRUCTIONS (MANDATORY)

1. Use **only arrays and collections** covered in syllabus.
2. Do **NOT** use LINQ.
3. Do **NOT** create additional classes.
4. Use **runtime input for all data**.
5. Perform validation where specified.
6. Output must be **clearly labeled**.
7. Do not hard-code values.
8. Program must execute without runtime errors.

TASK 1 – DYNAMIC PRODUCT PRICE ANALYSIS (ARRAYS)

WHAT THE STUDENT MUST DO

1. Ask the user to enter the **number of products**.
2. Create an integer array of that size.
3. Accept **only positive prices** for each product.
4. Calculate the **average price**.
5. Sort the array using `System.Array.Sort()`.
6. Replace prices **below average** with `0`.
7. Resize the array by **adding 5 new positions**.
8. Fill the newly added positions with the **average price**.
9. Display the final array with index positions.

EXPECTED OUTCOME

- Array is dynamically created.
- Prices below average are clearly marked as `0`.
- Array size increases correctly.
- Final array reflects sorting, replacement, and resizing.

TASK 2 – BRANCH SALES ANALYSIS (MULTI-DIMENSIONAL ARRAY)

WHAT THE STUDENT MUST DO

1. Ask the user to enter:
    - Number of branches
    - Number of months
2. Create a **2D integer array** accordingly.
3. Accept sales data for each branch and month.
4. Calculate:
    - Total sales per branch
    - Highest monthly sale across all branches
5. Display branch totals and global highest sale.

EXPECTED OUTCOME

- Correct 2D array creation.
- Accurate per-branch totals.
- Correct identification of highest sale value.

TASK 3 – PERFORMANCE-BASED DATA EXTRACTION (JAGGED ARRAY)

WHAT THE STUDENT MUST DO

1. Create a jagged array derived from the 2D sales array.
2. For each branch:
    - Include only sales values **greater than or equal to product average** (from Task 1).
3. Dynamically allocate each jagged row based on qualifying values.
4. Display jagged array contents branch-wise.

EXPECTED OUTCOME

- Jagged array sizes differ per branch.
- Only qualifying values appear.
- Demonstrates dependency between unrelated data structures.

TASK 4 – CUSTOMER TRANSACTION CLEANING (LIST & HASHSET)

WHAT THE STUDENT MUST DO

1. Ask for number of customer transactions.
2. Store customer IDs in a `List<int>` (duplicates allowed).
3. Convert the list into a `HashSet<int>` to remove duplicates.
4. Convert the HashSet back into a List.
5. Display:
    - Cleaned customer list
    - Number of duplicate entries removed

EXPECTED OUTCOME

- Duplicate customer IDs are removed.
- Original and cleaned data counts are correctly compared.

- Demonstrates round-trip conversion.

TASK 5 – FINANCIAL TRANSACTION FILTERING (DICTIONARY & SORTEDLIST)

WHAT THE STUDENT MUST DO

1. Ask for number of transactions.
2. Store transaction ID and amount in a `Dictionary<int, double>`.
3. Prevent duplicate transaction IDs.
4. Create a `SortedList<int, double>` containing only transactions:
    - Amount ≥ product average (from Task 1)
5. Display sorted high-value transactions.

EXPECTED OUTCOME

- Dictionary enforces unique keys.
- SortedList displays data in ascending key order.
- Cross-task filtering logic is applied correctly.

TASK 6 – PROCESS FLOW MANAGEMENT (STACK & QUEUE)

WHAT THE STUDENT MUST DO

1. Ask for number of operations.
2. Insert each operation into:
    - Queue (processing order)
    - Stack (undo tracking)
3. Process all queue elements in FIFO order.
4. Undo the **last two operations** using Stack.
5. Display processed and undone operations.

EXPECTED OUTCOME

- Queue processes tasks in correct order.
- Stack correctly reverses operations.
- Proper use of LIFO and FIFO behavior.

TASK 7 – LEGACY DATA RISK DEMONSTRATION (HASHTABLE & ARRAYLIST)

WHAT THE STUDENT MUST DO

1. Ask for number of users.
2. Store username-role pairs in a `Hashtable`.
3. Store the same data in an `ArrayList` in mixed form.
4. Display both collections.
5. Demonstrate why mixed-type storage is risky.

EXPECTED OUTCOME

- Hashtable displays key-value mapping.
- ArrayList contains mixed data types.
- Student shows understanding of legacy collection risks.

FINAL EXPECTATION FROM STUDENT

By completing all tasks, the student must demonstrate:

- Mastery of arrays and collections
- Runtime data handling
- Data dependency management
- Performance-aware design
- Correct selection of data structures
- Industry-level problem solving
