Faculty of Computers and Artificial Intelligence
Cairo University

Data Structure - Assignment #1

Assignment Instructions

- Use C++ only.
- Do not use the STL library.
- Ensure your code is efficient and releases all allocated memory properly.
- Any form of plagiarism will be detected and may result in a zero or **negative score**.
- You must understand every line of your code.
- All team members must be familiar with the assigned problems.
- Each team must consist of at least 4 and maximum 5 members.
- All team members must be from the same lab.
- The total score for the assignment is 100 points.
- Deadline: [28-3-2025].

Submission Instructions

- Each problem must be submitted in a separate `.cpp` file.
  Example: `Problem1.cpp`, `Problem2.cpp`, ...
- All `.cpp` files should be compressed into a single `.zip` file.
- The file name format should be as follows:
  (A1_ID1_ID2_ID3_ID4_<TA-NAME>.zip)
  Example: `A1_20220221_20220231_20220321_20220211_TA-Huda.zip`

Grading Notes

- If a specific step in the problem does not have a designated grade but is required, **missing it will result in a deduction from the total score.**
- Examples of such steps include:
  - Changing class names or function headers.
  - Modifying the sequence of inputs.
  - Altering the content of menus.

  Failure to follow these instructions may lead to a reduction in your final grade.

- Manual Test
  - Writing testcases consider as bonus (10 pt)
  - Use a file to read the input instead of “cin”

Faculty of Computers and Artificial Intelligence
Cairo University

Problem #1: Iftar Invitation Manager (15 pt)

Problem Statement:

Fatima loves hosting **Iftar gatherings** for friends and family. Since she invites different guests on different days, she needs a system to **track invitations**, including the guest's name, contact details, and the date they're invited.

Your task is to create an **object-oriented program** that:

1. **Stores** information about each guest invited for Iftar.
2. **Displays** a list of all invitations.
3. Allows Fatima to **update** the guest list if she adds or removes a guest.
4. Send a **reminder message** to guests on a specific date.
5. Write **testcase** for each part in the problem

Requirements:

1. Create a `Guest` class

Each guest should have the following attributes:

- `name` (string) → The guest's name (e.g., "Aisha").
- `contact` (string) → Their phone number or email.
- `iftar_date` (string) → The date they're invited (e.g., `"2025-03-15"`).

Include the following methods:

- `display_guest()` → Prints guest details.
- `update_invitation(new_date)` → Updates the guest's invitation date.

2. Create an `IftarManager` class

This class manages multiple invitations and has:o

- `guest_list` (list) → A list that stores multiple `Guest` objects.
- `add_guest(guest)` → Adds a new guest. **(2 pt)**
- `display_all_guests()` → Displays all invited guests. **(2 pt)**
- `update_guest_invitation(name, new_date)` → Updates a guest's invitation date. **(2 pt)**
- `send_reminder(date)` → Sends a reminder message to all guests on a specific date by email. **(5 pt)**

- `sort_guest_list()` → Sort guests by invitation date use any sorting algorithm. **(4 pt)**

Example Usage: In main() function

```
# Create Iftar Manager
manager = IftarManager()

# Add guests
guest1 = Guest("Aisha", "aisha@example.com", "2025-03-15")
guest2 = Guest("Omar", "omar@example.com", "2025-03-18")
guest3 = Guest("Zainab", "zainab@example.com", "2025-03-20")

manager.add_guest(guest1)
manager.add_guest(guest2)
manager.add_guest(guest3)

# Display guest list
manager.display_all_guests()

# Update invitation date for Omar
manager.update_guest_invitation("Omar", "2025-03-15")

# Display updated guest lis
manager.display_all_guests()

# Send reminders
manager.send_reminders()
```

Expected Output:

```
Guest: Aisha, Contact: aisha@example.com, Iftar Date: 2025-03-15
Guest: Omar, Contact: omar@example.com, Iftar Date: 2025-03-18
Guest: Zainab, Contact: zainab@example.com, Iftar Date: 2025-03-20

Updating invitation for Omar...

Guest: Aisha, Contact: aisha@example.com, Iftar Date: 2025-03-15
Guest: Omar, Contact: omar@example.com, Iftar Date: 2025-03-15
Guest: Zainab, Contact: zainab@example.com, Iftar Date: 2025-03-20
```

Faculty of Computers and Artificial Intelligence
Cairo University

Sending reminders...
Reminder sent to Aisha: Your Iftar invitation is on 2025-03-15!
Reminder sent to Omar: Your Iftar invitation is on 2025-03-15!

May your **Iftar gatherings** be full of warmth and blessings!

Problem #2 Polynomial Operations Using Pointers (10 pt)

Write a program that dynamically allocates arrays to store polynomials and performs operations on them. Given two polynomials, implement functions to:

1. **Display the polynomial** in a readable format. **(4 pt)**
2. **Calculate the sum** of two polynomials. **(3 pt)**
3. **Calculate the difference** (subtract the first from the second). **(3 pt)**
4. Write **testcase** for each part in the problem

Example 1

Input:

```
Order of first polynomial: 2
Enter polynomial: 0 1 3 2

Order of second polynomial: 4
Enter polynomial: 8 0 4 0 0 3
```

Output:

```
First polynomial: 2x^2 + 3x + 1 = 0
Second polynomial: 3x^4 + 4x = 8
Sum of polynomials: 3x^4 + 2x^2 + 7x + 1 = 8
Difference of polynomials: 3x^4 - 2x^2 + x - 1 = 8
```

Example 2

Input:

```
Order of first polynomial: 3
Enter polynomial: 0 5 0 1 4

Order of second polynomial: 3
Enter polynomial: 0 2 3 0 6
```

Output:

```
First polynomial: 4x^3 + x^2 + 5 = 0
Second polynomial: 6x^3 + 3x + 2 = 0
Sum of polynomials: 10x^3 + x^2 + 3x + 7 = 0
Difference of polynomials: 2x^3 - x^2 + 3x - 3 = 0
```

Problem #3 - Sorted Linked List (12 pt)

Implement a **Sorted Linked List (L)** that maintains its elements in sorted order using **insertion sort logic**. The list should support:

1. **Insertion (`insert(n)`)** → Adds n while keeping the list sorted. **(4 pt)**
2. **Deletion (`remove(n)`)** → Removes the element at index n (0-based index). If n is out of bounds, no changes occur. **(4 pt)**
3. **Overloaded Operators:**
    - **Output Operator (`<<`)** → Prints the linked list in a formatted way. **(2 pt)**
    - **Index Operator (`[ ]`)** → Returns the element at a given index. If out of bounds, an exception is thrown. **(2 pt)**
4. **Memory Management** → Ensure proper deallocation of memory when deleting nodes or destroying the list. **(-1 pt if didn't handle this point)**
5. Write **testcase** for each part in the problem

Example Test Cases

Test Case 1: Inserting Elements into the Sorted Linked List

Operations & Expected Output:

```
L.insert(5);   // L = [5]
L.insert(8);   // L = [5, 8]
```

```
L.insert(7);   // L = [5, 7, 8]
L.insert(6);   // L = [5, 6, 7, 8]
L.insert(6);   // L = [5, 6, 6, 7, 8]
cout << L;     // Output: [5, 6, 6, 7, 8]
```

Test Case 2: Accessing Elements Using Index Operator

Operations & Expected Output:

```
cout << L[2];  // Output: 6
cout << L[10]; // Throws out_of_range exception
```

Test Case 3: Deleting Elements from the Linked List

Starting List:

```
L = [5, 6, 6, 7, 8];
```

Operations & Expected Output:

```
L.remove(0);   // L = [6, 6, 7, 8]
cout << L;     // Output: [6, 6, 7, 8]

L.remove(100); // No change (out of bounds)
cout << L;     // Output: [6, 6, 7, 8]

L.remove(2);   // L = [6, 6, 8]
cout << L;     // Output: [6, 6, 8]

L.remove(2);   // L = [6, 6]
cout << L;     // Output: [6, 6]
```

Faculty of Computers and Artificial Intelligence
Cairo University

C++ Class Headers

```
#include <iostream>
using namespace std;

// Node class for linked list
class Node {
    public:
      int data;
      Node* next;
      Node(int val); // Constructor
};

// Sorted Linked List Class
class SortedLinkedList {
private:
    Node* head; // Pointer to the head of the list

public:
    SortedLinkedList();  // Constructor

    void insert(int value);  // Insert a value while keeping the list sorted
    void remove(int index);  // Delete node at given index

    // Overloaded operators
    friend ostream& operator<<(ostream& os, const SortedLinkedList& list);
    int operator[](int index);

    ~SortedLinkedList(); // Destructor to free memory
};
```

Problem #4 - Sorting Algorithms (55 pt)

Problem Statement:

The **Sorting System** is designed to help users **sort dynamically allocated data** using **nine different sorting algorithms**. It provides an **interactive menu** for selecting a sorting method, supports **various data types using templates**, and **displays each sorting step for better understanding**. Additionally, it **measures execution time** to analyze efficiency.

Requirements:

1. **Sorting Algorithms to Implement:**
    - **(1) Insertion Sort (4 pt)**

- **(2) Selection Sort (4 pt)**
- **(3) Bubble Sort (4 pt)**
- **(4) Shell Sort (4 pt)**
- **(5) Merge Sort (4 pt)**
- **(6) Quick Sort (4 pt)**
- **(7) Count Sort** *(only for integers)* **(4 pt)**
- **(8) Radix Sort** *(only for integers)* **(4 pt)**
- **(9) Bucket Sort (4 pt)**

2. **Interactive Menu: (5 pt)**
   - The program will display a **menu** listing all **nine sorting algorithms** and allow the user to **select the desired sorting method** by entering a number from **1 to 9**.
   - The user provides the **size of the dataset** and enters values dynamically.
3. **Template-Based Implementation: (5 pt)**
   - Sorting algorithms should be implemented using *templates (T)** to support **multiple data types** (int, float, double, string, etc.).
4. **Sorting Process Display: (9 pt)**
   - After every iteration of the sorting algorithm, **print the current state of the array** to help visualize the sorting process.
   - Use for each algorithm right visualization example for qsort print the selected pivot with array state like the following example

```
Data: 78 34 12 90 50 60

Pivot: 60 → [34, 12, 50]  60  [78, 90]
Pivot: 12 → [12]  34  50  60  [78, 90]
Pivot: 90 → [78]  90
```

5. **Performance Tracking:**
   - Measure and **display execution time** for each sorting algorithm.
6. Write **testcase** for each part in the problem

Class Design - `SortingSystem` (Header)

```
#include <iostream>
using namespace std;

template <typename T>
class SortingSystem {
private:
    T* data;  // Dynamic array for storing input data
    int size; // Size of the array

public:
    SortingSystem(int n);  // Constructor
    ~SortingSystem();      // Destructor

    void insertionSort();  // (1) Insertion Sort
    void selectionSort();  // (2) Selection Sort
    void bubbleSort();     // (3) Bubble Sort
    void shellSort();      // (4) Shell Sort
    void mergeSort(int left, int right); // (5) Merge Sort
    void quickSort(int left, int right); // (6) Quick Sort
    void countSort();      // (7) Count Sort (Only for int)
    void radixSort();      // (8) Radix Sort (Only for int)
    void bucketSort();     // (9) Bucket Sort

    void merge(int left, int mid, int right); // Merge Sort Helper
    int partition(int low, int high); // Quick Sort Helper

    void displayData();  // Print the current state of the array
    void measureSortTime(void (SortingSystem::*sortFunc)()); // Measure sorting time

    void showMenu(); // Display menu for user interaction
};
```

Faculty of Computers and Artificial Intelligence
Cairo University

Full Test Case - Sorting System (Selection Sort on Palestinian Cities)

Step 1: User Enters Data Size

```
Enter the number of items to sort: 9
```

Step 2: User Enters Data to Sort (Palestinian Cities in Arabic Spelling - English Letters)

```
Enter data 1: Nablus
Enter data 2: Gaza
Enter data 3: Al-Khalil
Enter data 4: Ramallah
Enter data 5: Ariha
Enter data 6: Jenin
Enter data 7: Tolkarem
Enter data 8: Al-Quds
Enter data 9: Yafa
```

Step 3: Display Sorting Menu

```
Select a sorting algorithm:
1. Insertion Sort
2. Selection Sort
3. Bubble Sort
4. Shell Sort
5. Merge Sort
6. Quick Sort
7. Count Sort (Only for integers)
8. Radix Sort (Only for integers)
9. Bucket Sort
Enter your choice (1-9): 2
```

Step 4: Selection Sort Execution (Step by Step)

```
Sorting using Selection Sort...
Initial Data: [Nablus, Gaza, Al-Khalil, Ramallah, Ariha, Jenin,
Tolkarem, Al-Quds, Yafa]
```

```
Iteration 1: [Al-Khalil, Gaza, Nablus, Ramallah, Ariha, Jenin,
Tolkarem, Al-Quds, Yafa]
Iteration 2: [Al-Khalil, Al-Quds, Nablus, Ramallah, Ariha, Jenin,
Tolkarem, Gaza, Yafa]
Iteration 3: [Al-Khalil, Al-Quds, Ariha, Ramallah, Nablus, Jenin,
Tolkarem, Gaza, Yafa]
Iteration 4: [Al-Khalil, Al-Quds, Ariha, Gaza, Nablus, Jenin,
Tolkarem, Ramallah, Yafa]
Iteration 5: [Al-Khalil, Al-Quds, Ariha, Gaza, Jenin, Nablus,
Tolkarem, Ramallah, Yafa]
Iteration 6: [Al-Khalil, Al-Quds, Ariha, Gaza, Jenin, Nablus,
Tolkarem, Ramallah, Yafa]
Iteration 7: [Al-Khalil, Al-Quds, Ariha, Gaza, Jenin, Nablus,
Ramallah, Tolkarem, Yafa]
Iteration 8: [Al-Khalil, Al-Quds, Ariha, Gaza, Jenin, Nablus,
Ramallah, Tolkarem, Yafa]

Sorted Data: [Al-Khalil, Al-Quds, Ariha, Gaza, Jenin, Nablus,
Ramallah, Tolkarem, Yafa]
Sorting Time: 0.00018 seconds
```

Step 5: Ask User If They Want to Sort Again

```
Do you want to sort another dataset? (y/n): n
Thank you for using the sorting system! Goodbye!
```

Problem #5 - Statistical Calculation (8 pt)

Problem Statement:

You are given a sequence of numbers. Your task is to do the following and write **testcase** for each part in the problem:

Calculate basic statistical values, including:

- **Median** (Middle value of the sorted sequence) **(2 pt)**
- **Minimum** (Smallest number in the sequence) **(1 pt)**

- **Maximum** (Largest number in the sequence) **(1 pt)**
- **Mean** (Average of all numbers) **(2 pt)**
- **Summation** (Total sum of all numbers) **(2 pt)**

The program will allow the user to:

- Enter a sequence of numbers dynamically.
- Choose which statistical operation they want to perform from a **specific statistics menu**.

Class Design - `StatisticalCalculation` (Header)

```
#include <iostream>
template <typename T>

class StatisticalCalculation {

     private:
          T* data;            // Dynamically allocated array for storing data
          int size;           // Number of elements in the array

     public:
          StatisticalCalculation(int size);
          ~StatisticalCalculation();

          void sort();   // Implement Sort Algorithm
          // Statistical Calculation Functions
          T findMedian();
          T findMin();
          T findMax();
          double findMean();
          T findSummation();

          // Utility Functions
          void displayArray();  // Display sorted array
          void inputData();     // Take input dynamically
          void statisticsMenu(); // Menu for statistical operations

     };
```

Program Flow & Menu System

Step 1: User Inputs the Sequence of Numbers

```
Enter the number of elements: 7

Enter element 1: 9
```

```
Enter element 2: 3

Enter element 3: 4

Enter element 4: 66

Enter element 5: 7

Enter element 6: 6

Enter element 7: 8
```

Step 2: Statistics Menu Selection

```
Select a statistical calculation:

  1. Find Median

  2. Find Minimum

  3. Find Maximum

  4. Find Mean

  5. Find Summation

  Enter your choice (1-5): 1
```

Step 3: Calculating and Displaying the Result

```
    Median: 7
```