Object Oriented Programming (Fall 2023)

BS CS F22

Assignment-01

General Instructions:

- **Deadline for this assignment submission is Wednesday 6th of March 2024.**
- You must zip your files into a folder and name it according to your roll no., for example**,** **A01_BCSF22M001.zip**. This zipped file should then be submitted via email: **progfun15@gmail.com**.
- This is an individual assignment. Any case of cheating or plagiarism would result in a solid **zero** marks.

Task 1: Implementing a Matrix Abstract Data Type (ADT)

Objective: *To implement a Matrix ADT in C++ that represents a 2D dynamically allocated matrix and provides various operations such as inputting values, displaying the matrix, transposing, adding scalar and another matrix, and more.*

Instructions:

- Create a C++ class named **Matrix** that represents a matrix and encapsulates its data and operations.
- The class should have the following private data members:
    - **rows**: an integer representing the number of rows in the matrix.
    - **cols**: an integer representing the number of columns in the matrix.
    - **data**: a 2D array of integers to store the matrix elements dynamically allocated using pointers.
- Implement a constructor to initialize the matrix with the given number of rows and columns. Make sure to dynamically allocate memory for the matrix data.
- Don't forget to implement appropriate copy constructor.
- Implement a destructor to deallocate the dynamically allocated memory for the matrix data.
- Implement the following public member functions (methods) for the **Matrix** class:
    - **inputValues()**: Method to input values into the matrix from the user.
    - **setValues()**: Method to set values of calling object same as values of object received as parameter.
    - **getValue(int row, int col)**: Returns the value at position defined by row and col
    - **setValue(int row, int col, int value)**: Sets the value at position defined by row and col
    - **show()**: Method to display the matrix in matrix form.
    - **transpose()**: Method to transpose the matrix. It will return an object.
    - **addScalar(int scalar)**: Method to add a scalar value to all elements of the matrix.
    - **addMatrix(const Matrix& other)**: Method to add another matrix to the current matrix.
    - **subtractScalar(int scalar)**: Method to subtract a scalar value from all elements of the matrix.
    - **subtractMatrix(const Matrix& other)**: Method to subtract another matrix from the current matrix.
    - **multiplyScalar(int scalar)**: Method to multiply a scalar value with all elements of the matrix.
    - **mutiplyMatrix(const Matrix& other):** First check, if compatible, multiply two matrices according to matrix multiplication rules and return the new matrix as object.
    - **determinant()**: Method to calculate determinant of the matrix. (for now, you can implement only for 2x2). Implementing determinant of a 3x3 matrix will be entitle you for some bonus points.
    - **adjoint()**: Method to find adjoint of the matrix. (for now, you can implement only for 2x2)
    - **inverse()**: Method to find inverse of the matrix. (for now, you can implement only for 2x2)
    - **isSymmetric()**: A boolean method to check if a matrix is symmetric or not.

Task 2: Implementing a String Abstract Data Type (ADT)

Objective: *To implement a String ADT in C++ that represents a variable-length string and provides various operations such as inputting values, displaying the string, getting and setting characters, concatenation, substring extraction, copying strings, copy-constructor, converting to uppercase, lowercase, proper case, sentence case, and more.*

Task 2 - Instructions:

Create a C++ class named MyString that represents a string and encapsulates its data and operations.

- The class should have the following private data members:
  - **length:** an integer representing the length of the string.
  - **data:** a dynamically allocated character array to store the string.
- Implement a constructor to initialize the string with an empty value. Make sure to dynamically allocate memory for the string data.
- Implement a destructor to deallocate the dynamically allocated memory for the string data.
- Implement a copy constructor to initialize a string using another string by copying its data. This involves deep copying of the character array to ensure that each object has its own separate copy of the string data.
- Implement the following public member functions (methods) for the **MyString** class:
  - **inputValues()**: Method to input values into the string from the user.
  - **getValue(int index)**: Returns the character at the specified index in the string
  - **setValue(int index, char value)**: Sets the character at the specified index in the string.
  - **show()**: Method to display the string.
  - **concatenate(const MyString& other)**: Method to concatenate another string to the current string.
  - **substring(int start, int length)**: Method to extract a substring from the string starting at the specified index with the given length.
  - **copyString(const MyString& other):** Method to copy another string to the current string.
  - **toUpper()**: Method to convert the string to uppercase.
  - **toLower():** Method to convert the string to lowercase.
  - **toProper():** Method to convert the string to proper case (capitalize the first letter of each word).
  - **toSentence():** Method to convert the string to sentence case (capitalize the first letter of the string).
  - **Sentence():** Returns sentence case of the string (but do not modify calling object).
  - **toReverse():** Change the string with reverse of the string.
  - **Reverse():** Return reverse of the string (but do not modify calling object).

- Test your implementation by creating multiple instances of both the classes and performing various operations on them. Do include your driver function in the .zip file.

Note:

- **Implement getters and setters for all attributes of the class.**
- **Provide input validation wherever required.**
- Make sure to handle memory management properly, including dynamic memory allocation and deallocation.
- Provide comments in your code to explain the functionality of each method.
- Clearly mention the naming convention you use.
- Test your implementation thoroughly to ensure correctness and robustness.