Project Title: Task Management System

Project Description:

In this project, I will design and implement a Task Management System that utilizes stacks, queues, and object-oriented programming (OOP) principles in C++.
The goal is to create an application that can manage tasks efficiently, providing features like adding tasks, viewing tasks, adjusting priority, and completing tasks.

Objectives

1. Understand Data Structures: Gain practical experience with stacks and queues by implementing them in the context of task management.
2. Learn Object-Oriented Principles: Apply OOP principles such as encapsulation, inheritance, and polymorphism to create a well-structured application.
3. Develop Problem-Solving Skills: Use data structures to solve a practical problem, reinforcing logical thinking and programming skills.

Tools used:

1. Class Design:

• Create a Task class that represents a single task with attributes such as name, description, priority, and status (e.g., pending, completed).
• Create a TaskManager class that manages a list of tasks. This class should utilize both a stack
and a queue.
– Use a stack to handle tasks marked with high priority (last in, first out).
– Use a queue to manage normal tasks (first in, first out).

2. Functionality:

• Use the TaskManager class to implement the following methods:
– add task(task): Add a new task to the appropriate data structure based on its priority.

– view tasks(): Display all current tasks, including their status.
– complete task(): Mark the next task as complete based on priority.
– remove completed(): Remove all completed tasks from the system.
– increase priority(task id): Move a task from the queue to the stack to give it high priority.

3. User Interface:

• Implement a simple command-line interface that allows users to interact with the task management system through prompts.

• Allow users to add tasks, view tasks, mark tasks as completed and change task priorities via text input.

4. Testing:

• Run a series of test cases to validate the behavior of the TaskManager methods.

Deliverables
• A C++ script containing the implementation of the Task and TaskManager classes.
• A simple command-line interface that allows interaction with the system.
• Main function that illustrates the code with an example.

Example Output
Task Management System

Commands:
1. Add a task
2. View tasks
3. Complete a task
4. Increase priority
5. Remove completed tasks
6. Exit
Initial State:

Priority Stack (High Priority):
[Empty]
Normal Queue (Normal Priority):
[1] "Clean the house" (Pending, Priority: Low)
[2] "Submit assignment" (Pending, Priority: Normal)
Example Interaction:

> Add a task
Enter task name: Buy groceries
Enter task description: Milk, Eggs, Bread
Enter task priority (High/Normal/Low): High
-> Task 'Buy groceries' added successfully to the priority stack!
Updated State:

Priority Stack (High Priority):
[1] "Buy groceries" (Pending, Priority: High)
Normal Queue (Normal Priority):
[1] "Clean the house" (Pending, Priority: Low)
[2] "Submit assignment" (Pending, Priority: Normal)
> View tasks
Current Tasks:
1. "Buy groceries" (Pending, Priority: High)

2. "Clean the house" (Pending, Priority: Low)
3. "Submit assignment" (Pending, Priority: Normal)

> Complete a task
-> Task 'Buy groceries' marked as completed!
Updated State:

Priority Stack (High Priority):
[Empty]
Normal Queue (Normal Priority):
[1] "Clean the house" (Pending, Priority: Low)
[2] "Submit assignment" (Pending, Priority: Normal)
> Remove completed tasks
-> Completed tasks removed successfully.
> View tasks
Current Tasks:
1. "Clean the house" (Pending, Priority: Low)
2. "Submit assignment" (Pending, Priority: Normal)
> Increase priority
Enter task ID to increase priority: 1
-> Task 'Clean the house' moved to high priority.
Updated State:

Priority Stack (High Priority):
[1] "Clean the house" (Pending, Priority: High)
Normal Queue (Normal Priority):
[1] "Submit assignment" (Pending, Priority: Normal)
> Exit
Thank you for using the Task Management System!

Criteria
• Correct implementation of stacks and queues.
• Adherence to OOP principles.
• Usability and functionality of the command-line interface.