**Task Manager Assignment - Design Document (Due date: Feb 14, 2025 by 3:00 p.m.)**

## **Student Information**

* **Name: Jesse Melgar**
* **Student ID: 2020151717**
* **Course: CMPS2131**
* **Date: Feb 13, 2025**

## **Project Overview**

Students are to create a console application Task Manager. A task Manager application helps users manage and organize tasks. Users should be allowed to use CRUD (Create/Add, Read, Update & Delete) operations. The program will use a priority based singly linked list, which will show 1 of 3 priorities: High, Medium, Low. Students are to create a menu driven interface in which Users can easily interact with the system.

## **Data Structures**

### **Task Class/Struct**

* Attributes:
  + Description: std::string
  + Priority: enum or string
  + Due Date: std::string
  + Completion: Bool
  + Next Pointer: Task \*
* Justification: These attributes are essential for the task management system. Description provides task details, priority organizes based on….. Priority… self explanatory id say… due date reminds users about tasks, ensuring timely completion. Completion status shows the user whether the task is completed (T) or not (F). Next pointer facilitates and enables linked list nav.

### **TaskList Class**

* **How will the linked list be managed?** (Head/Tail pointers, insertion logic)
  + The linked list will be managed using head & tail pointers (respectively, head points to the first task & tail points to the last)
  + As for the insertion logic, tasks will be inserted based on their priority.
  + In regards to Ordering, tasks will be ordered primarily by their priority.

## **Core Functionalities**

### **Task Management**

* How will tasks be **added**, **removed**, and **marked as complete**?
  + Add (Create): Users can input task details (description, priority, due date) through the provided interface.
  + Mark as Complete (Update): User will mark as complete and task will no longer be displayed as active.
  + Remove (Delete) User will search the task by description, delete its node from the linked list and update the previous & next pointers to maintain integrity and functionality.
* How will tasks be **searched** by description?

### **Display Operations**

* How will tasks be **displayed** and **filtered**?
  + The system will display all uncompleted tasks (by default, in order of priority)
  + Users will be able to filter through tasks by priority
* How will the system handle showing tasks due within **N days**?
  + The Task Manager will calculate the difference between current and due date.

## **Error Handling & Edge Cases**

Memory Allocation Error: Provide clear message to the user and use try-catch blocks for critical operations.

Invalid Input: Validate user input (Priority shouldnt be an issue in this case but date format must be entered correctly. Users will be provided with the format (dd/mm/yyyy) minimizing the error margin.)

## **Memory Management Plan**

Memory leaks: Ensuring that all allocated memory is deallocated using delete.

Dynamis Memory: Use new to allocate memory for tasks. Use delete to free memory when the program shuts down & when the tasks are removed

## 

## **User Interface Plan**

Once users start the program, they will be prompted into a text menu which would have this view & structure:

| 1. | Add Task |
| --- | --- |
| 2. | Remove Completed Task |
| 3. | Mark a Task as Complete |
| 4. | Search Tasks |
| 5. | Display Tasks |
| 6. | Filter by Priority |
| 7. | Show Task due within N days |
| 8. | Exit |

As for the inputs, users will be prompted to insert with cin>>

## **Testing Strategy**

After each option is coded, Student will test alongside 1 other person:

Adding tasks, Searching for task, removing task, Displaying Task due within N days.

Each will be tested one by one. Once the entire program is done, further testing will be done: mix with each other to ensure there are no issues.

For example, If user adds a task then removes it, obviously it shouldnt appear when displaying all tasks. The same goes for Marking as complete. It should not display as active. Should there be any errors, student will document the error, what happened and how it was fixed.

## **Additional Features (if any)**

Should student complete program before due date, additional features will be added, such as (but not limited to):

Sorting tasks by due date, Saving task to a file then loading it up when program starts.

## 

## **Approval (Instructor Use Only)**

* **Approved:** Yes / No
* **Instructor Comments:**