Shaheed Zulfikar Ali Bhutto Institute vof Science & Technology University

DEPARTMENT OF ROBOTICS & ARTIFICIAL INTELLIGENCE

Total Marks: 04

Obtained Marks: ________

Data Structure

&

Algorithms

Final Project Report

Last date of Submission: 20th, December, 2025

Submitted To: **Mr. Annas Khalid Khan**

Student Name	Registration Number
Muhammad Danish Wahab	24108244
Shehryar Ali	2410266
Arsalan Ahmed	24108114
Tajjuddin Khan	24108234

Smart Task Scheduler - Project Report

Contents

1. Project Overview

The **Smart Task Scheduler** is a comprehensive C++ console-based application designed to help users manage, organize, and schedule tasks efficiently using fundamental data structures and algorithms. This system implements real-world task management concepts with priority-based scheduling, conflict detection, and undo functionality.

Key Features: Task creation, update, and deletion with unique IDs. Priority-based task queuing (1=highest to 5=lowest). Automatic and manual scheduling algorithms. Conflict detection between scheduled tasks. Complete history tracking and undo operations. Multiple display views for different task states. Data persistence through runtime memory management.

Technical Stack: Programming Language: C++. Data Structures: Linked Lists, Queues, Stacks, Arrays. Algorithms: Bubble Sort, Recursive Search, Conflict Detection. Concepts: Object-Oriented Programming, Memory Management, Recursion.

2. Project Scope

2.1 Functional Scope

Included Functionalities:

- Task Management Module: Create new tasks with name, priority, deadline, and duration; update existing task details; delete tasks from the system; unique ID generation for each task. Scheduling Module: Automatic scheduling based on priority and deadline; manual task selection for scheduling; conflict detection algorithm; separation of scheduled and pending tasks. Display Module: View all tasks in the system; display scheduled tasks; show pending tasks due to conflicts; view scheduling history; monitor priority queues. Utility Module: Undo last scheduling operation; input validation and error handling; clean memory management; user-friendly console interface.

2.2 Technical Scope

Core Components:

- Task Class - Central data structure for task representation. Linked Lists - For storing all tasks, scheduled tasks, and pending tasks. Priority Queues - For organizing tasks by priority levels. Undo Stack - For implementing undo functionality. History Tracking - Linked list for scheduling history. Sorting Algorithm - Bubble sort for task prioritization. Recursive Functions - For conflict checking and display operations.

2.3 Limitations

- No File Storage - All data is lost when program closes. No GUI - Console-based interface only. No Multi-user Support - Single-user system. Basic Conflict Detection - Simple priority and deadline-based conflicts. Fixed Array Sizes - Limited stack and queue capacities.

3. Module Distribution & Team Contribution

Team Structure & Responsibilities

Team Member	Role	Responsibilities	Technical Modules
Muhammad Danish Whab	**Project Manager & Lead Developer**	Overall project coordination, architecture design, main algorithm implementation, integration testing	TaskManager class, Main program flow, Scheduling algorithms, System integration
Tjjuddin Khan	**Data Structures Specialist**	Core data structure implementation, memory management, optimization	Task class, Linked Lists, History system, Bubble sort algorithm
Shehryar Khan	**User Interface & Operations**	User interaction modules, display systems, input validation	Display functions, Menu system, Input validation, Queue operations
Arsalan Ahmad	**Utility & Support Modules**	Supporting functionality, utility classes, error handling	UndoStack class, PriorityQueue class, Helper functions, Conflict detection

3.1 Detailed Module Breakdown

Module 1: Task Management System (Muhammad Danish Wahab)

- Task Class Implementation: Task attributes and data members; constructor and initialization; display function for task details; deep copy functionality for task duplication. Input Validation System: Integer validation with range checking; string input handling; buffer clearing operations.

Module 2: Core Data Structures (Tjjuddin Khan)

- Linked List Implementation: Main task list management; scheduled tasks list; pending tasks list; history linked list with HistoryNode class. Sorting Algorithm: Bubble sort implementation; priority-based sorting (primary); deadline-based sorting (secondary). Recursive Functions: Recursive task display; recursive conflict checking.

Module 3: User Interface & Display (Shehryar Khan)

- Menu System: Main menu display; navigation flow control; user choice handling. Display Functions: All tasks display; scheduled tasks view; pending tasks view; history display; queue status display. Queue Management: PriorityQueue class implementation; enqueue and dequeue operations; queue status display.

Module 4: Utility & Support (Arsalan Ahmad)

- Undo System: UndoStack class implementation; stack operations (push/pop); task restoration functionality. Utility Operations: Task search by ID; task removal from lists; memory cleanup operations.

Module 5: Scheduling Engine (Muhammad Danish Whab)

- Automatic Scheduling: Priority-based task selection; conflict detection algorithm; task distribution between scheduled and pending. Manual Scheduling: User-selected task scheduling; individual conflict checking; history recording. System Integration: Connecting all modules; ensuring data flow consistency; error handling and edge cases.

3.2 Development Timeline

- Week 1-2: Planning & Design - Requirement analysis; architecture design; module distribution; data structure planning. Week 3-4: Core Implementation - Task class and basic structures; linked list implementation; input/output systems. Week 5-6: Algorithm Development - Sorting algorithm; scheduling logic; conflict detection. Week 7-8: User Interface - Menu system; display functions; user interaction modules. Week 9-10: Integration & Testing - Module integration; bug fixing; performance optimization; final testing.

4. Conclusion

4.1 Project Achievements

- The Smart Task Scheduler successfully demonstrates the practical application of data structures and algorithms in solving real-world problems. Key achievements include: Effective Task Management: Comprehensive system for task creation, modification, and deletion. Intelligent Scheduling: Priority-based algorithms that mimic real-world task prioritization. Robust Data Structures: Efficient use of linked lists, queues, and stacks. User-Friendly Interface: Intuitive menu system with clear feedback. Reliable Performance: Stable operation with proper memory management.

4.2 Learning Outcomes

- **Technical Skills Developed:** Advanced C++ programming techniques. Implementation of complex data structures. Algorithm design and optimization. Memory management and pointer operations. Recursive function implementation. Object-oriented design principles.
- **Team Collaboration Benefits:** Effective module-based development. Clear responsibility distribution. Regular integration and testing. Problem-solving through team discussion. Version control and code management.

4.3 Future Enhancements

- Database Integration - Add file storage for data persistence. Advanced Scheduling - Implement calendar-based scheduling. GUI Development - Create graphical user interface. Network Capability - Add multi-user support. Mobile Application - Develop cross-platform version. AI Integration - Implement machine learning for smart scheduling. Reporting System - Generate task completion reports.

4.4 Final Remarks

- The Smart Task Scheduler project successfully demonstrates how theoretical computer science concepts can be applied to create practical, useful applications. Through effective teamwork and clear module distribution, the team created a robust system that not only meets the specified requirements but also provides a foundation for future enhancements.

- ✓ The project highlights the importance of: Proper planning and design. Clear module boundaries. Regular testing and integration. Effective team communication. Documentation and code quality.

- ✓ This project serves as an excellent example of collaborative software development, showcasing how individual contributions combine to create a cohesive, functional system that exceeds the sum of its parts.

==============================