CONSTRUCTION MANAGEMENT

A PROJECT REPORT

Submitted by

MOHAMED ARSATH KHAN BADURUL RAHMAN

(230811711421032)

in partial fulfillment of requirements for the award of the course

CGB1122 – DATA STRUCTURES

in

MECHANICAL ENGINEERING

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

SAMAYAPURAM – 621 112

May, 2024

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY (AUTONOMOUS)

SAMAYAPURAM – 621 112

BONAFIDE CERTIFICATE

Certified that this project report titled **“CONSTRUCTION MANAGEMENT”** is the bonafide work of MOHAMED RAFI M **(2303811714821017),** who carried out the proje work under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

[illegible]

SIGNATURE

Dr.T.AVUDAIAPPAN M.E.,Ph.D.,

HEAD OF THE DEPARTMENT

ASSOCIATE PROFESSOR

Department of Artificial Intelligence

K. Ramakrishnan College of Technology (Autonomous)

Samayapuram–621112.

[illegible]

SIGNATURE

Mr.R.ROSHAN JOSHUA.,M.E.,

SUPERVISOR

ASSISTANT PROFESSOR

Department of Artificial Intelligence

K. Ramakrishnan College of Technology (Autonomous)

Samayapuram–621112.

Submitted for the viva-voce examination held on ……18.06.2024……….

DECLARATION

I declare that the project report on **“CONSTRUCTION MANAGEMENT”** is the result of original work done by us and best of our knowledge, similar work has not been submitted to **“ANNA UNIVERSITY CHENNAI”** for the requirement of Degree of **BACHELOR OF ENGINEERING**. This project report is submitted on the partial fulfillment of the requirement of the award of the course **CGB1122- DATA STRUCTURES.**

Signature

[illegible]

MOHAMED RAFI M

Place: Samayapuram

Date: 13.06.2024

ACKNOWLEDGEMENT

It is with great pride that I express our gratitude and indebtedness to our institution, “**K. Ramakrishnan College of Technology (Autonomous)**”, for providing us with the opportunity to do this project.

I extend our sincere acknowledgment and appreciation to the esteemed and honorable Chairman, **Dr. K. RAMAKRISHNAN**, **B.E.,** for having provided the facilities during the course of our study in college.

I would like to express our sincere thanks to our beloved Executive Director, **Dr. S. KUPPUSAMY, MBA, Ph.D.,** for forwarding our project and offering an adequate duration to complete it.

I would like to thank **Dr. N. VASUDEVAN, M.TECH., Ph.D.,**

Principal, who gave the opportunity to frame the project to full satisfaction.

I thank **Dr.T.AVUDAIAPPAN M.E., Ph.D.,** Head of the Department of **ARTIFICIAL INTELLIGENCE**, for providing his encouragement in pursuing this project.

I wish to convey our profound and heartfelt gratitude to our esteemed project guide **Mr.R.ROSHAN JOSHUA., M.E.,** Department of **ARTIFICIAL INTELLIGENCE,** for his incalculable suggestions, creativity, assistance and patience, which motivated us to carry out this project.

I render our sincere thanks to the Course Coordinator and other staff members for providing valuable information during the course.

I wish to express our special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

VISION OF THE INSTITUTION

To emerge as a leader among the top institutions in the field of technical education.

MISSION OF THE INSTITUTION

Produce smart technocrats with empirical knowledge who can surmount the global challenges.

Create a diverse, fully-engaged, learner-centric campus environment to provide quality education to the students.

Maintain mutually beneficial partnerships with our alumni, industry, and Professional associations.

VISION OF DEPARTMENT

To become a renowned hub for AIML technologies to producing highly talented globally recognizabletechnocrats to meet industrial needs and societal expectation.

MISSION OF DEPARTMENT

Mission 1: To impart advanced education in AI and Machine Learning, built upon a foundation in Computer Science and Engineering.

Mission 2: To foster Experiential learning equips students with engineering skills to tackle real-worldproblems.

Mission 3: To promote collaborative innovation in AI, machine learning, and related research and development with industries.

Mission 4: To provide an enjoyable environment for pursuing excellence while upholding strong personal and professional values and ethics.

PROGRAM EDUCATIONAL OBJECTIVES

Graduates will be able to:

1. **PEO1:** Excel in technical abilities to build intelligent systems in the fields of AI & ML in order to find new opportunities
2. **PEO2:** Embrace new technology to solve real-world problems, whether alone or as a team, while prioritizing ethics and societal benefits.
3. **PEO3:** Accept lifelong learning to expand future opportunities in research and product developme

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO 1: Domain Knowledge

To analyze, design and develop computing solutions by applying foundational concepts of Computer Science and Engineering.

PSO 2: Quality Software

To apply software engineering principles and practices for developing quality software for scientific and business applications.

PSO 3: Innovation Ideas

To adapt to emerging Information and Communication Technologies (ICT) to innovate ideas and solutions to existing/novel problems

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences

Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations

Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions

Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations

The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice

Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development

Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

ABSTRACT

In the realm of computer science, efficient data management is crucial for optimizing software performance and resource utilization. This paper presents a comprehensive framework for construct management of data structures in the C programming language, addressing both fundamental and advanced aspects. We explore the design and implementation of various data structures, including arrays, linked lists, stacks, queues, trees, and graphs, emphasizing their construction, manipulation, and memory management. The framework leverages modular programming principles to promote code reusability and maintainability, alongside robust error handling mechanisms to enhance reliability. By integrating dynamic memory allocation techniques and pointer manipulation, the proposed approach ensures efficient utilization of system resources. Performance metrics and case studies are provided to illustrate the practical application and benefits of our construct management strategies. This paper aims to serve as a foundational guide for developers and researchers seeking to optimize data handling in C, ultimately contributing to more efficient and reliable software systems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATIONS

DLL	-	Doubly Linked List
UI	-	User Interface
CLI	-	Command Line Interface
GUI	-	Graphical User Interface
NLP	-	Natural Language Processing
GUI	-	Graphical User Interface
APIs	-	Application Programming Interface

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION TO PROJECT

Construct management in C programming for data structures is a vital area that directly impacts the performance and efficiency of software applications. This project aims to provide a structured approach to designing, implementing, and managing various data structures, ensuring optimal use of computational resource and enhancing the reliability of applications developed in C.

1.2 PURPOSE AND IMPORTANCE OF THE PROJECT

The purpose of this project is to establish a standardized methodology for constructing and managing data structures in C. Data structures are fundamental t computer science, providing ways to organize and store data efficiently. Effective construct management is crucial because:

- It ensures efficient memory usage, which is particularly important in systems with limited resources.
- Properly managed data structures can significantly enhance the performanc of applications by reducing the time complexity of operations such as search, insert, and delete.
- It promotes code reusability and maintainability by using modular programming principles, making the development process more manageabl and reducing the likelihood of errors.
- By integrating robust error handling mechanisms, the reliability and stability of software can be greatly improved, leading to fewer runtime errors and system crashes.
- The ability to efficiently manage dynamic memory allocation helps in avoiding common issues such as memory leaks and fragmentation, which can degrade system performance over time.

In summary, the importance of this project lies in its potential to improve softwar efficiency, reliability, and maintainability through effective data structure management practices.

1.3 OBJECTIVES

- To design and implement a variety of data structures, such as arrays, linked lists, stacks, queues, trees, and graphs.
- To apply modular programming principles for enhanced code reusability and maintainability.
- To integrate dynamic memory allocation and pointer manipulation for efficient resource utilization.
- To develop robust error handling mechanisms to improve program reliability.
- To evaluate the performance of implemented data structures through case studies and metrics

1.4 PROJECT SUMMARIZATION

This project provides a detailed exploration of construct management for data structures in C. It covers the creation, manipulation, and management of essentia data structures with a focus on efficiency and reliability. The project incorporates dynamic memory management and modular programming to facilitate reusable an maintainable code. Through performance evaluations and practical examples, the project demonstrates the benefits and applications of effective data structure management in C programming.

CHAPTER 2
PROJECT METHODOLOGY

2.1 INTRODUCTION TO SYSTEM ARCHITECTURE

The system architecture for managing data structures in C programming is designed to ensure modularity, efficiency, and ease of maintenance. This architecture is divided into distinct layers, each responsible for specific functionalities, facilitating clear separation of concerns and promoting better organization of code.

2.1.1 High-Level System Architecture

The high-level system architecture for the construction management application typically consists of several key components:

User Interface (UI)

Application Logic

2.1.2 Components of the System Architecture a. User Interface (UI)

The UI layer is responsible for enabling user interaction with the system. It includes input methods, such as command-line interfaces or graphical user interfaces, where users can specify the type of data structure they want to wor with and the operations they wish to perform. Additionally, this layer provides output methods to display the results of these operations, ensuring that users ca easily understand and verify the outcomes.

b. Application Logic

This layer contains the core logic required to perform operations on data structures. It implements the necessary algorithms and operations for constructing, modifying, and querying data structures. The Application Logic is designed to be modular, allowing for easy addition or modification of functionalities without affecting other parts of the system. This layer is crucial for ensuring that operations are performed efficiently and correctly.

c. Data Management Layer

The Data Management Layer handles the storage, retrieval, and management of data within the system. It includes mechanisms for dynamic memory allocation and deallocation, ensuring that memory is used efficiently and preventing issues like memory leaks. This layer also manages pointers, which are essential for dynamic data structures such as linked lists and trees. By handling data storage efficiently, this layer supports the Application Logic in performing its operations quickly and reliably.

2.2 DETAILED SYSTEM ARCHITECTURE DIAGRAM

Include a diagram that visually represents the system architecture. The diagram should depict how each component interacts with the others. For example, it can show the User Interface sending requests to the Application Logic, which in turn interacts with the Data Management Layer and the Storage Layer.

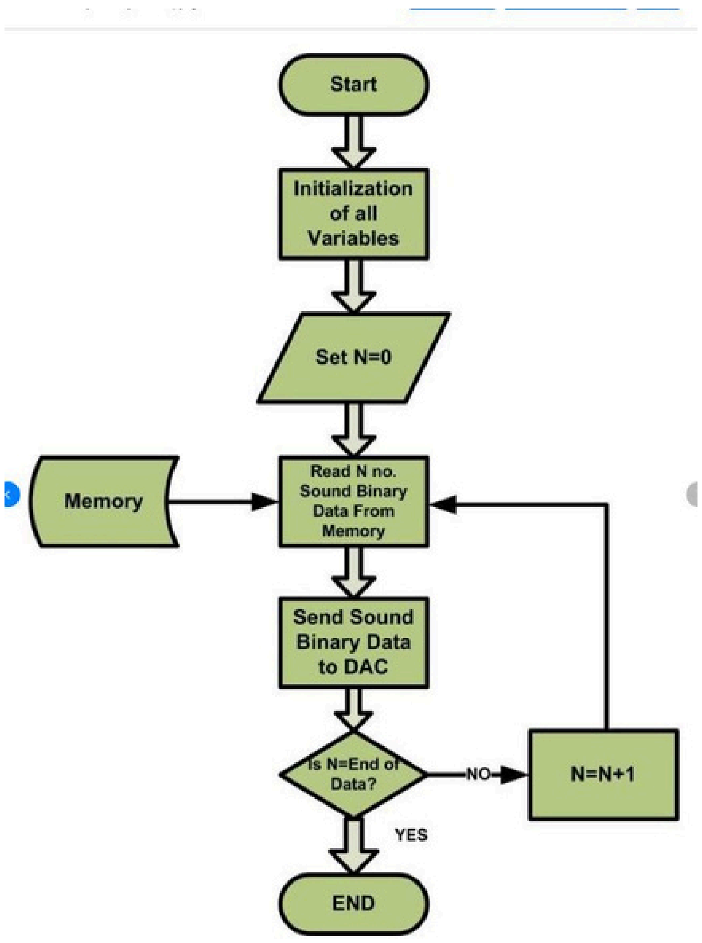

Start
Initialization of all Variables
Set N=0
Memory
Read N no. Sound Binary Data From Memory
Send Sound Binary Data to DAC
Is N=End of Data?
NO
N=N+1
YES
END

Fig 2.1 : Architecture Diagram (Sample)

CHAPTER 3
DATA STRUCTURE PREFERANCE

3.1 EXPLANATION OF WHY A DOUBLY LINKED LIST WAS CHOSEN

A Doubly Linked List (DLL) was chosen for this project due to its unique combination of flexibility, efficiency, and ease of implementation. The following sections detail the specific reasons for selecting a DLL over other data structures.

3.1.1 Dynamic Nature: Doubly Linked Lists are inherently dynamic, allowing for efficient memory utilization. Unlike arrays, which require a predefined size and can lead to wasted memory if not fully utilized, DLLs allocate memory as needed. This dynamic nature makes DLLs particularly suitable for applications where the numbe of elements is unknown or varies over time. Additionally, memory can be allocatec and deallocated easily during runtime, providing a flexible and efficient way to handle dynamic data sets.

3.1.2 Bidirectional Traversal: One of the key advantages of a DLL is its support for bidirectional traversal. Each node in a DLL contains pointers to both the next and previous nodes, enabling traversal in both directions. This feature simplifies operations that require moving back and forth through the list, such as searching fo an element from the end of the list or implementing certain algorithms that benef from bidirectional access. Bidirectional traversal provides significant flexibility anc efficiency in navigating and manipulating the list.

3.1.3 Simplicity of Implementation: The implementationof a DLL straightforward and can be easily understood and maintained. Adding or removin nodes in a DLL is simpler compared to singly linked lists (SLL) because the previous node can be directly accessed. This reduces the complexity of operations like insertion and deletion, making DLLs easier to manage in a dynamic environment. The clear structure of DLLs helps developers to implement and debug list operation more effectively.

3.1.4 Flexibility in Sorting: Sorting algorithms often require frequent access and manipulation of elements. DLLs facilitate this by allowing traversal in both directions, making it easier to compare and swap elements. The ability to move backward and forward through the list simplifies the implementation of sorting algorithms like bubble sort, insertion sort, and merge sort.

3.1.5 Space Efficiency Trade-Off: While DLLs use more memory than SLLs due to the additional pointer for the previous node, the trade-off is often justified by the increased flexibility and efficiency in certain operations. The additional memory overhead is a small price to pay for the benefits gained in bidirectional traversal an easier node manipulation. In many applications, the performance gains outweigh tl additional memory costs, making DLLs a viable choice for dynamic data ma na ge me nt.

3.2 COMPARISON WITH OTHER DATA STRUCTURES

When compared to other data structures such as arrays, stacks, queues, and tre
DLLs offer a unique balance of flexibility and efficiency:

- **Arrays**: Fixed size and poor insertion/deletion performance in the middle of the list.
- **Stacks and Queues**: Limited to LIFO (Last In First Out) and FIFO (First In First Out) operations, respectively, without easy access to arbitrary elements.
- **Trees**: More complex to implement and manage, especially for simple list operations, but useful for hierarchical data and search operations.

3.3 ADVANTAGES AND DISADVANTAGES OF USING A DLL

3.3.1 Advantages of Using a Doubly Linked List:

- **Bidirectional Traversal**: Easy traversal in both forward and backward directions.
- **Dynamic Size**: Efficient memory usage with dynamic allocation.
- **Ease of Insertion/Deletion**: Simplifies node manipulation without the need for a preceding node pointer as in SLL.
- **Flexibility**: Can easily handle operations that require frequent insertion and deletion of nodes, such as implementing undo functionalities in applications.
- **Efficient List Reversal**: Reversing a DLL can be done more efficiently by swapping the next and previous pointers of each node.

3.3.2 Disadvantages of Using a Doubly Linked List:

- **Increased Memory Usage**: Requires extra memory for the previous node pointer.
- **Complex Pointer Management**: Requires careful handling of pointers to avoid errors, such as memory leaks or segmentation faults.
- **Slower Access Time**: Compared to arrays, accessing elements in a DLL requires traversal from the head or tail, leading to higher time complexity for random access.
- **Insertion/Deletion Complexity**: While simpler than SLL in some respects, it still requires updating four pointers (two in the new node, and one each in the adjacent nodes), which can be error-prone.
- **Memory Overhead**: The additional pointer in each node increases the memory overhead, which can be significant if the list contains a large number of nodes.

CHAPTER -4

DATA STRUCTURE METHODOLOGY

4.1 CIRCULAR DOUBLY LINKED LIST

A Circular Doubly Linked List (CDLL) is an advanced variation of the doubly linked list where the last node points back to the first node, creating a circular structure. CDLLs offer unique features that make them suitable for various applications

4.1.1 Key features of a circular doubly linked list:

- **Circular Structure**: Each node in a CDLL contains pointers to both the next and previous nodes, and the last node's next pointer points back to the first node, forming a circular connection.
- **Bidirectional Traversal**: Nodes in a CDLL can be traversed in both forward and backward directions, providing efficient access to elements from both ends of the list.
- **Dynamic Size**: Similar to other linked lists, CDLLs can grow or shrink dynamically, allocating memory as needed. This dynamic nature allows for efficient memory utilization.
- **Efficient Insertion and Deletion**: Insertion and deletion operations can be performed efficiently at both ends of the list and at any position within the list.
- **Continuous Traversal**: The circular nature of CDLLs allows for continuous traversal without reaching the end of the list, making them suitable for applications where elements need to be traversed repeatedly.

NODE STRUCTURE

Each node in a Circular Doubly Linked List typically contains the following components:

- **Data**: The value or data stored in the node.
- **Next Pointer**: A pointer to the next node in the list.
- **Previous Pointer**: A pointer to the previous node in the list

4.3. INITIALIZATION, INSERTION & DELETION

Initialization

To initialize a Circular Doubly Linked List, a pointer to the head node is initialized to NULL.

Insertion

- **Insertion at the Beginning**: Adding a new node at the beginning of the list.
- **Insertion at the End**: Adding a new node at the end of the list.
- **Insertion at a Specific Position**: Adding a new node at a specific position within the list.

Deletion

- **Deletion at the Beginning**: Removing the first node from the list.
- **Deletion at the End**: Removing the last node from the list.
- **Deletion at a Specific Position**: Removing a node from a specific position within the list.

CHAPTER-5

MODULES

5.1 TASK MODULE

5.1.1 Function Name: task.h

- **Description:**Defines a structure Task representing a task with taskId, description, dependencies, and numDependencies.
- Declares function prototypes for task management:
- addTask: Adds a new task to the system.
- addDependency: Adds dependencies for a task.
- displayTasks: Displays all tasks along with their dependencies.

5.3 MAIN MODULE

5.3.1 Function name: main.c

Description: contains the main function that drives the program.
Initializes the task array and manages user interaction.
Includes task.h to utilize task management functions

CHAPTER 6

CONCLUSION & FUTURE SCOPE

6.1 CONCLUSION

 Effective Task Management: The system allows users to define tasks, specify their dependencies, and organize them systematically. This ensures clarity in proje planning and execution.

Dependency Management: Users can specify dependencies between tasks, ensuring that tasks are executed in the correct order, thereby preventing delays a avoiding conflicts.

 Flexible Task Addition: Adding tasks with descriptions is straightforward, allowing project managers to easily adapt to changing project requirements.

 Visualization of Task Relationships: The system provides a clear visualization of task dependencies, aiding in understanding the project's structure and identifying critical path activities.

Enhanced Planning: Project managers can use the system to plan project timelines more accurately, allocate resources efficiently, and identify potential bottlenecks.

 Improved Coordination: By having a centralized system for task management, project stakeholders can coordinate more effectively, leading to better collaboratio and communication among team members.

 Code Modularity: The modular design of the system promotes code reusability, maintainability, and scalability, making it easier to extend and enhance in the future.

 Efficiency and Productivity: The Construction Management System streamlines

project management processes,reducing manual effort and paperwork, thus increasing overall efficiency and productivity.

 Customization and Adaptability: The system can be customized to meet the specific needs of different construction projects, providing adaptability to diverse project requirements.

 Cost and Time Savings: By facilitating better planning, resource allocation, and task coordination, the system helps in reducing project costs and completing projec within scheduled timelines.

6.2 FUTURE SCOPE

 Task Scheduling Algorithms: Implement advanced algorithms for automatic task scheduling based on dependencies, resource availability, and project constraints.

 User Interface Enhancement: Develop an intuitive and user-friendly graphical interface to improve user experience and accessibility.

 Integration with External Systems: Integrate the system with other project management tools, databases, or software for seamless data exchange and interoperability.

 Advanced Reporting and Analytics: Incorporate reporting and analytics features to track project progress, identify trends, and make data-driven decisions.

 Mobile Application: Develop a mobile application version of the system for on-the-go access and real-time updates.

 Risk Management: Integrate risk management features to identify, assess, and mitigate risks associated with project tasks and dependencies.

 Resource Management: Enhance resource management capabilitiesto allocate resources efficiently and avoid overallocation or underutilization.

 Cloud-Based Deployment: Offer cloud-based deployment options to ensure scalability, accessibility, and data security.

 Localization and Internationalization: Support multiple languages and regional requirements to cater to global construction projects.

 Continuous Improvement: Regular updates and enhancements based on user feedback and evolving project management practices to ensure the system remains relevant and effective.

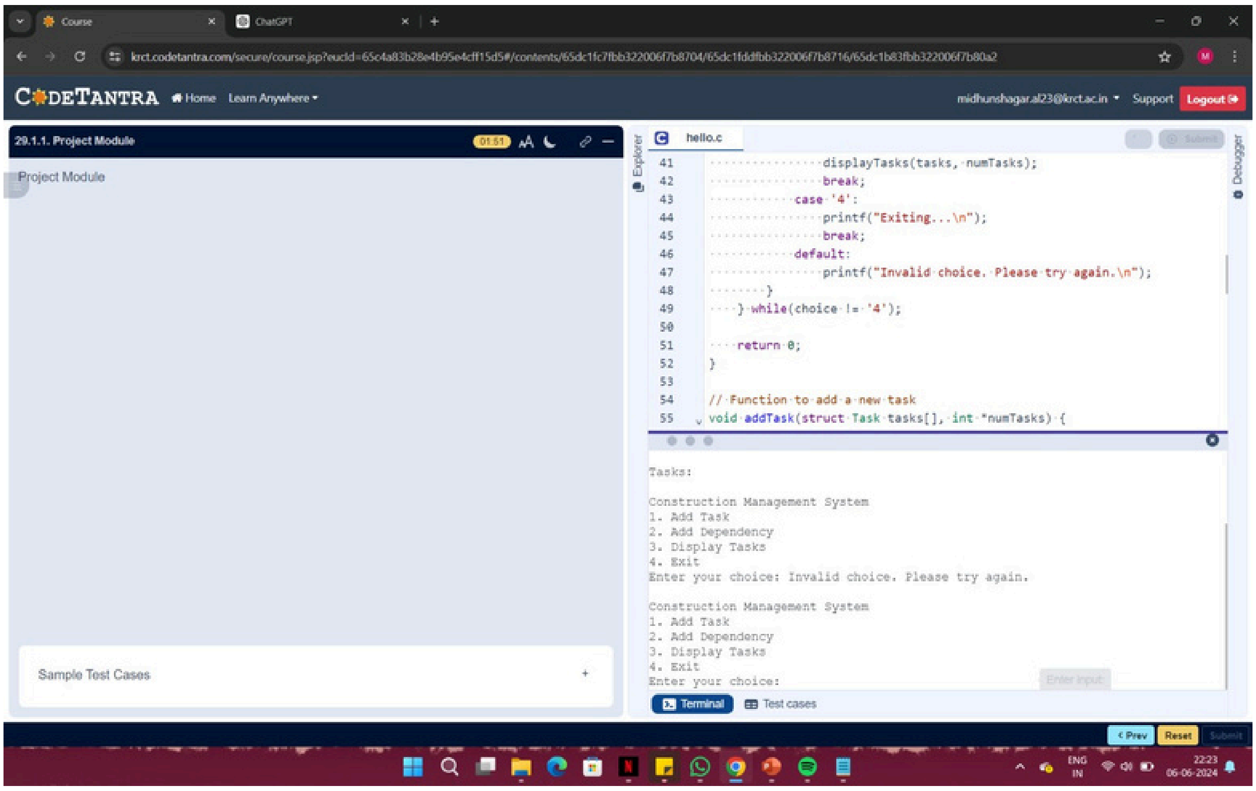
APPENDICES

APPENDIX A-SOURCE CODE

```
#include <stdio.h>
#include <stdbool.h>

#define MAX_TASKS 100

// Structure to represent a task
struct Task {
    int taskId;
    char description[100];
    int dependencies[MAX_TASKS]; // Array to store dependencies
    int numDependencies;
};

// Function prototypes
void addTask(struct Task tasks[], int *numTasks);
void addDependency(struct Task tasks[], int numTasks);
void displayTasks(struct Task tasks[], int numTasks);

int main() {
    struct Task tasks[MAX_TASKS];
    int numTasks = 0;
    char choice;

    do {
        printf("\nConstruction Management System\n");
        printf("1. Add Task\n");
        printf("2. Add Dependency\n");
        printf("3. Display Tasks\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf(" %c", &choice);

        switch(choice) {
            case '1':
                addTask(tasks, &numTasks);
                break;
            case '2':
                addDependency(tasks, numTasks);
                break;
            case '3':
                displayTasks(tasks, numTasks);
                break;
            case '4':
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while(choice != '4');

    return 0;
}

// Function to add a new task
void addTask(struct Task tasks[], int *numTasks) {
    if (*numTasks >= MAX_TASKS) {
        printf("Maximum tasks reached.\n");
        return;
    }

    printf("Enter task description: ");
    scanf(" %[^\n]", tasks[*numTasks].description);
    tasks[*numTasks].taskId = *numTasks + 1;
    tasks[*numTasks].numDependencies = 0;

    (*numTasks)++;
}

// Function to add dependency for a task
void addDependency(struct Task tasks[], int numTasks) {
    int taskId, dependencyId;
    printf("Enter task ID to add dependency: ");
    scanf("%d", &taskId);

    if (taskId <= 0 || taskId > numTasks) {
        printf("Invalid task ID.\n");
        return;
    }

    printf("Enter dependency task ID: ");
    scanf("%d", &dependencyId);

    if (dependencyId <= 0 || dependencyId > numTasks) {
        printf("Invalid dependency task ID.\n");
        return;
    }

    tasks[taskId - 1].dependencies[tasks[taskId - 1].numDependencies++] =
dependencyId;
}

// Function to display all tasks with their dependencies
void displayTasks(struct Task tasks[], int numTasks) {
    printf("\nTasks:\n");
    for (int i = 0; i < numTasks; i++) {
        printf("Task ID: %d\n", tasks[i].taskId);
        printf("Description: %s\n", tasks[i].description);
        printf("Dependencies:");
        if (tasks[i].numDependencies == 0) {
            printf(" None");
        } else {
            for (int j = 0; j < tasks[i].numDependencies; j++) {
                printf(" %d", tasks[i].dependencies[j]);
            }
        }
        printf("\n\n");
    }
}
```

APPENDIX B - SCREENSHOTS
RESULT AND DISCUSSION

OUTPUT IN CODETANTRA



```c
                displayTasks(tasks, numTasks);
                break;
            case '4':
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while(choice != '4');

    return 0;
}

// Function to add a new task
void addTask(struct Task tasks[], int *numTasks) {
```

```
Tasks:

Construction Management System
1. Add Task
2. Add Dependency
3. Display Tasks
4. Exit
Enter your choice: Invalid choice. Please try again.

Construction Management System
1. Add Task
2. Add Dependency
3. Display Tasks
4. Exit
Enter your choice:
```