**QUIZ GAME**

**USING DATASTRUCTURES**

*A Project Report submitted to*

**JNTUA, Ananthapuramu**

In partial fulfilment of the requirements for the award of the degree of

# Bachelor of Technology

(Computer Science & Engineering) By

**B. PARDHA MANOJ REDDY (24KB1A0538)**

**B. VIJAY KUMAR (24KB1A0555)**

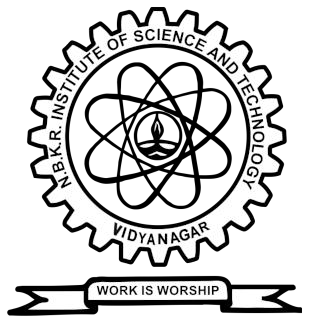
**B. HAJARATHBABU (24KB1A0571)**

**C. SANDEEP (24KB1A0579)**

Under the esteemed Guidance of

**Sivanraj, B.Tech., M. Tech., [Ph. D]**

## Assistant Professor, Department of CSE



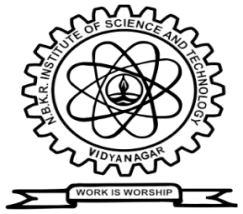
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

**N.B.K.R. INSTITUTE OF SCIENCE & TECHNOLOGY**

**(AUTONOMOUS)**

**VIDYANAGAR, TIRUPATI (DIST), AP– 524413**

**APRIL 2025**



|  |  |
| --- | --- |
| Website: www.nbkrist.org. | Ph: 08624-228 247 |
| Email: ist@nbkrist.org. | Fax :08624-228 257 |

**N.B.K.R. INSTITUTE OF SCIENCE & TECHNOLOGY**

**(Autonomous)**

**(Approved by AICTE: Accredited by NBA: Affiliated to JNTUA, Ananthapuramu) An ISO**

**9001-2000 Certified Institution**

**Vidyanagar -524 413, Tirupati District, Andhra Pradesh, India**

# BONAFIDECERTIFICATE

This is to certify that the project work entitled “**Quiz Game**” is a bonafide work done by **B. Pardha Manoj Reddy (24KB1A0538), B. Vijay Kumar (24KB1A0555), B. Hajarathbabu (24KB1A0571), C. Sandeep (24KB1A0579)** in the department of **Computer Science & Engineering, N.B.K.R. Institute of Science & Technology**, **Vidyanagar** and is submitted to **JNTUA, Ananthapuramu** in the partial fulfillment for the award of B. Tech degree in **Computer Science & Engineering.** This work has been carried out under my supervision.

|  |  |
| --- | --- |
| ***Sivanraj*** | ***Dr. A. Rajasekhar Reddy*** |
| *Trainer (Byte XL),* | *Professor & Head* |
| *Department of CSE,* | *Department of CSE,* |
| *NBKRIST, Vidyanagar.* | *NBKRIST, Vidyanagar.* |

Submitted for the Viva-Voce Examination held on \_\_\_\_\_\_\_\_\_\_\_

**Examiner-1 Examiner-2**

## ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of a project would be incomplete without the people who made it possible of their consistent guidance and encouragement crowned our efforts with success.

We would like to express our profound sense of gratitude to our project guide **Sivanraj Trainer (Byte XL), Department of Computer Science and Engineering, N.B.K.R. Institute of Science and Technology (Affiliated to JNTUA, Ananthapuramu)**, Vidyanagar, for his masterful guidance and the constant encouragement throughout the project. Our sincere appreciations for his suggestions and unmatched services without which this work would have been an unfulfilled dream.

We convey our special thanks to **Dr. Y. VENKATARAMI REDDY, Chairman,**

**N.B.K.R. Institute of Science and Technology,** for providing excellent infrastructure in our campus for the completion of the project.

We convey our special thanks to **Sri. N. RAM KUMAR, Correspondent** of **N.B.K.R. Institute of Science and Technology**, for providing excellent infrastructure in our campus for the completion of the project.

We are grateful to **Dr. V. VIJAYA KUMAR REDDY, Director, N.B.K.R. Institute of Science and Technology** for allowing me to avail all the facilities in the college.

We express our sincere gratitude to **Dr. A. RAJASEKHAR REDDY, Professor& Head, Department of Computer Science and Engineering**, for providing exceptional facilities for successful completion of my project.

We would like to convey our heart full thanks to **Staff Members, Lab Technicians, and my friends**, who extend their cooperation in making this project as a successful one.

We would like to thank one and all who have helped me directly and indirectly to complete this project successfully.

### ABSTRACT

The Data Structures Quiz Game is an interactive, command-line based educational game designed to test and enhance a player's understanding of fundamental concepts in Data Structures. This project is developed in the C programming language, utilizing arrays for storing a pool of questions and a linked list for recording the player's answers.

The quiz is divided into three difficulty levels: Easy, Medium, and Hard.

Each level consists of 10 carefully crafted questions related to various data structures like arrays, stacks, queues, linked lists, trees, heaps, graphs, and sorting algorithms.

From each level, 5 random questions are selected during the game to ensure variability and to challenge the player's knowledge.

To proceed to the next difficulty level, the player must score at least 3 out of 5 correct answers. Failure to meet this qualification ends the game.

This dynamic progression system motivates players to understand and perform better at each stage.

**The primary objectives of this project are:**

Reinforce knowledge of data structures through practical quizzing.

Demonstrate the use of arrays and linked lists in C for real-world applications.

Incorporate basic randomization and dynamic memory management concepts.

Develop logical reasoning and quick decision-making under time constraints (optional future scope).

This project serves as a strong foundation for beginners to strengthen their programming logic, data structure fundamentals, and C language proficiency.

## LIST OF FIGURES

### FIG NO FIGURE NAME PAGE NO

3.1.1 System pipeline 10

3.2.1 Methodology 11

4.2.1 System Architecture 15

4.3.1 Data Flow Diagram 26

6.2.1 Input &Output 35

## TABLE OF CONTENTS

**CHAPTER NO CHAPTER NAME PAGENO**

**ACKNOWLEDGEMENT**

**ABSTRACT**

**LIST OF FIGURES**

**CHAPTER 1 INTRODUCTION 1-4**

1.1 Introduction

1.2 Background and Motivation

1.3 Problem Statement

1.4 Objectives and Scope

1.5 Organization of the Project Report

1.6 Summary

**CHAPTER 2 SURVEY OF LITERATURE 5-9**

2.1 Introduction

2.2 Literature Survey

2.3 Identification of Research Gap

2.4 Existing System

2.5 Proposed System

2.6 Feasibility Analysis

2.7 Summary

**CHAPTER 3 METHODOLOGY 10-12**

3.1 Introduction

3.2 Overview of Methodological Approach

3.3 Description of Tools and Technologies Used

|  |  |  |
| --- | --- | --- |
| **CHAPTER 4** | **SYSTEM DESIGN**  4.1 Introduction  4.2 Detailed Design of Components  4.3 Data Flow Diagram  4.4 UML Diagrams  4.5 System Requirements  4.6 Summary | **13-20** |
| **CHAPTER 5** | **IMPLEMENTATION**  5.1 Introduction  5.2 Code Structure and Organization Procedure  5.3 Algorithms and Techniques Implemented  5.4 Summary | **21-33** |
| **CHAPTER 6** | **RESULTS AND ANALYSIS**  6.1 Introduction  6.2 Results  6.3 Analysis of Performance Metrics  6.4 Discussion of Findings  6.5 Summary | **34-37** |
| **CHAPTER 7** | **CONCLUSION AND FUTURE WORK** | **38-39** |

7.1 Conclusions Drawn from the Study

7.2 Limitations and Challenges Encountered

7.3 Suggestions for Future Work

**CHAPTER 8 REFERENCES 40**

List of References Sited in the Report

**CHAPTER–1**

**INTRODUCTION**

### OVERVIEW

The rapid evolution of technology and the growing demand for competent software developers have placed significant importance on mastering fundamental computer science concepts. Among these, data structures form a critical foundation. Understanding data structures is essential for designing efficient algorithms and building high-performance applications. However, for many students, grasping these abstract concepts through traditional lectures and textbooks alone can be challenging.

To bridge this gap between theory and practice, this project introduces the Data Structures Quiz Game, an interactive and educational application developed in the C programming language. The core purpose of this system is to allow students and learners to reinforce their understanding of various data structure concepts through a dynamic, quiz-based format. The game provides an engaging way to test knowledge across different difficulty levels, while also demonstrating how data structures like arrays and linked lists can be implemented in real-life applications.

The game is structured into three difficulty levels: Easy, Medium, and Hard. Each level contains a bank of ten carefully curated multiple-choice questions related to fundamental and advanced data structures such as arrays, stacks, queues, linked lists, trees, heaps, and graphs. From each level, five random questions are presented to the user, who must answer at least three correctly to proceed to the next stage. This scoring and progression system not only makes the quiz interactive but also ensures that users are assessed progressively based on their knowledge level.

From a technical standpoint, the game demonstrates effective use of arrays for storing questions and linked lists for dynamically storing the user's answers. This design choice highlights the contrast between static and dynamic memory allocation, both of which are key topics in data structure courses. Additionally, the use of randomization functions in C ensures that each playthrough is unique, enhancing the replayability and educational value of the system.

The user interface is built using a simple command-line interface (CLI), making it platform-independent and easy to run on any system with a C compiler. Despite its simplicity, the interface is intuitive and user-friendly, guiding the user through each stage of the quiz with clear instructions and feedback.

Overall, the Data Structures Quiz Game is more than just a project—it is a practical tool for learning, revision, and self-assessment. It allows students to engage with the subject matter in a way that is interactive, enjoyable, and directly tied to the real implementation of core programming concepts. This overview outlines the motivation and significance behind the project, setting the stage for the detailed chapters that follow in this report.

### 1.2 BACKGROUND AND MOTIVATION

Understanding data structures is crucial in computer science, as they play a key role in solving problems efficiently. However, many learners struggle to grasp these concepts due to their complexity. To address this, interactive tools like quizzes can enhance learning by reinforcing knowledge in an engaging way.

The motivation behind this quiz game is to provide an interactive method for learners, especially beginners, to strengthen their understanding of data structures. The game is designed to:

Enhance Learning: It tests both theoretical and practical knowledge, reinforcing key concepts.

Challenge and Progress: Players advance through difficulty levels, encouraging growth and mastery.

Immediate Feedback: Instant corrections help learners identify and learn from mistakes.

Gamification: By making learning fun, it keeps students motivated and engaged.

Reinforce Critical Thinking: The game encourages problem-solving, which is essential for coding interviews and technical assessments..

### 1.3 PROBLEM STATEMENT

### The problem at hand is the difficulty students face in mastering data structures due to their abstract nature and the complexity involved in understanding their real-world applications. Traditional learning methods, such as lectures and textbooks, often fail to engage students effectively, leading to a lack of retention and understanding. Furthermore, applying these concepts in problem-solving scenarios can be daunting for beginners.

### To address this, there is a need for an interactive learning tool that combines education with engagement. A quiz-based game focusing on data structures can serve as an effective solution, allowing students to test and reinforce their knowledge in a fun and competitive environment. This game should include varying levels of difficulty, immediate feedback, and a scoring system to motivate students to progress and improve.

### 1.4 OBJECTIVES & SCOPE

**OBJECTIVES:**

* **The objectives of the data structures quiz game are:**
* **Enhance Knowledge Retention:** Provide an interactive platform for students to test and reinforce their understanding of data structures, leading to better retention of key concepts.
* **Motivate Progressive Learning:** Offer varying levels of difficulty (easy, medium, hard) to challenge learners and encourage them to progressively master more complex topics.
* **Immediate Feedback and Learning:** Allow instant feedback on answers to help learners identify mistakes, understand the correct answers, and improve their understanding in real-time.
* **Promote Critical Thinking:** Engage players in problem-solving, reinforcing their ability to think critically and apply data structure concepts effectively in real-world scenarios.
* **Improve Engagement through Gamification:** Make learning fun and motivating by using a game format, which enhances student engagement and encourages continued participation.
* **Prepare for Technical Assessments:** Provide practice for coding interviews and technical assessments by offering questions that simulate real-world problem-solving scenarios.
* **Track Progress:** Keep track of player scores and answers, allowing users to measure their progress and identify areas that need improvement.

**SCOPE:**

**The scope of the data structures quiz game includes:**

**Coverage of Key Data Structures:** The game will cover essential data structures like arrays, stacks, queues, trees, and graphs, with varying difficulty levels (easy, medium, hard).

**Interactive Learning and Feedback:** Players will receive real-time feedback on their answers, helping to reinforce learning and improve understanding.

**Gamification and Progression:** The game will use a scoring system and levels to keep players engaged, encouraging continuous improvement and preparation for technical assessments.

### 1.5 ORGANIZATION OF THE PROJECT REPORT

The Data Structures Quiz Game is designed to enhance students' understanding of data structures through an engaging and interactive approach. The project aims to address the common challenge of mastering complex concepts by providing a platform for learners to test their knowledge in a fun, game-like environment. The game covers key data structures such as arrays, stacks, queues, trees, and graphs, offering questions at different difficulty levels. By incorporating immediate feedback and a progression system, the game motivates students to improve their skills continuously. It also helps learners develop critical thinking and problem-solving abilities, which are essential for real-world applications and technical interviews. The project is built using a user-friendly interface, ensuring accessibility across multiple platforms. Through gamification, the game maintains student engagement and encourages retention of key concepts. Ultimately, this tool provides a dynamic way for students to learn and practice data structures effectively.

### 1.6 SUMMARY

The Data Structures Quiz Game is an interactive tool designed to help students learn and reinforce their understanding of key data structures like arrays, stacks, queues, trees, and graphs. The game offers questions at varying difficulty levels (easy, medium, and hard) and provides immediate feedback to enhance learning. It aims to engage students through gamification, motivating them to progress through levels and improve their scores. By testing both theoretical knowledge and practical application, the game fosters critical thinking and prepares learners for real-world technical challenges, such as coding interviews. Ultimately, the quiz game serves as an effective and enjoyable way to strengthen knowledge of data structures.

**CHAPTER–2**

**SURVEY OF LITERATURE**

### 2.1 INTRODUCTION

The study of data structures is a cornerstone of computer science education, forming the foundation for efficient algorithm design and problem-solving. Over the years, various methods and tools have been developed to teach and reinforce concepts related to data structures. Traditional educational approaches, such as textbooks and lectures, while effective, often fail to engage students fully or facilitate active learning. This has led to the exploration of more interactive and engaging methods, such as computer-based tutorials, quizzes, and gamified learning environments, which have proven to be more effective in reinforcing complex concepts.

Literature on this topic reveals a growing interest in integrating technology into the learning process. Research suggests that interactive tools, like quizzes and games, significantly improve student engagement, retention, and understanding of abstract topics such as data structures. These methods not only make learning enjoyable but also encourage continuous assessment and self-paced learning. Various studies have demonstrated that gamification, when applied correctly, can enhance motivation, promote problem-solving, and aid in the long-term retention of knowledge.

This section aims to explore existing work on interactive learning tools for data structures, focusing on the effectiveness of quiz-based games and other educational technologies. By reviewing current trends and methodologies, the objective is to identify gaps in the existing tools and explain how the proposed project aims to address these challenges.

**2.2 LITERATURE SURVEY:**

**1.** Several studies have explored the effectiveness of interactive learning tools for teaching data structures. Online quizzes and interactive exercises have been found to help students retain information and improve problem-solving abilities, particularly in complex areas like binary trees and sorting algorithms.

**2.** Gamification in education has been proven to increase student engagement and motivation. By incorporating game design elements like points, levels, and rewards, students are encouraged to continue learning and progress through challenges, leading to higher participation and better learning outcomes.

**3.** Educational games specifically designed for data structures, such as AlgoBot and Data Structures Game, have demonstrated the potential to teach fundamental concepts like stacks, queues, and trees. These games offer challenges and puzzles that require players to apply their knowledge to solve problems, fostering a deeper understanding of the material.

**4.** Online platforms like LeetCode, HackerRank, and GeeksforGeeks offer coding challenges and quizzes to help students practice data structures. These platforms provide a self-directed learning environment with real-time feedback, which helps solidify concepts and prepare students for technical interviews.

**5.** Despite the success of interactive tools, many existing platforms still lack personalized learning experiences, and often focus solely on either theory or coding practice, leaving students with a fragmented understanding of data structures.

**6.** Existing literature highlights the importance of combining both theoretical and practical knowledge in a gamified format, a feature that is often missing in many current learning tools.

**7.** The proposed quiz game aims to fill these gaps by offering a comprehensive, level-based approach that integrates both conceptual understanding and practical application of data structures, providing immediate feedback and incorporating elements of gamification to motivate learners.

**2.3 IDENTIFICATION OF RESEARCH GAP:**

**1. Lack of Integrated Learning Approaches:** While many existing tools focus on either theoretical knowledge or practical coding skills, few integrate both aspects effectively. Most platforms either emphasize theoretical quizzes or coding challenges separately, leaving a gap in tools that assess both theoretical understanding and practical application in a seamless manner.

**2. Personalization in Learning Tools:** Many interactive learning tools lack personalized learning experiences, which could help tailor the difficulty of questions or challenges based on the learner's current level of understanding. Existing platforms do not adjust dynamically to the individual’s progress, hindering a more customized and effective learning experience.

**3. Engagement and Retention in Data Structures Learning:** Although gamification has been shown to enhance engagement, many educational games and platforms do not fully leverage game mechanics like competition, rewards, and progress tracking to maintain long-term interest. This results in learners losing motivation and not retaining key concepts as effectively.

**4. Limited Depth in Theoretical Content:** Most interactive quiz games and coding platforms tend to focus on problem-solving skills and practical implementation, but they often fall short in reinforcing the underlying theoretical principles of data structures. A comprehensive tool that integrates both theory and practice in a balanced way is needed.

**5. Feedback Mechanisms:** While quizzes provide feedback, they often lack detailed explanations that can help students understand why certain answers are correct or incorrect. This gap in providing insightful feedback limits the learning experience, as students are unable to fully grasp the reasoning behind correct solutions.

**6. Lack of Multi-Level Difficulty Progression:** Many existing quiz platforms do not have a clear progression of difficulty levels, which can result in learners either feeling overwhelmed by difficult questions or bored with easy ones. There is a need for tools that dynamically adjust difficulty as learners advance.

### 2.4 EXISTING SYSTEM

Existing systems for learning data structures include online quizzes, coding platforms, gamified tools, and traditional textbook-based methods. While platforms like LeetCode, GeeksforGeeks, and code signal focus on coding challenges, they primarily address practical skills without a deep emphasis on theoretical understanding. Gamified tools like AlgoBot engage learners through interactive challenges, but they often lack comprehensive theoretical explanations. Textbook-based learning provides a strong theoretical foundation but lacks interactivity and engagement. Although some platforms offer automated feedback, it tends to be superficial, leaving gaps in understanding.

**Disadvantages:**

### Limited integration of both theoretical knowledge and practical coding skills.

### Lack of personalization in adjusting difficulty based on learner progress.

### Superficial feedback without in-depth explanations for incorrect answers.

### Insufficient engagement, especially in non-gamified or static systems.

### 2.5 PROPOSED SYSTEM

The proposed system is an interactive, level-based quiz game designed to teach data structures through a combination of theoretical questions, coding challenges, and gamification. It integrates both conceptual understanding and practical application, allowing students to test their knowledge in an engaging and comprehensive manner. The game is structured in three difficulty levels: Easy, Medium, and Hard, with each level offering progressively challenging questions. The system offers immediate feedback, explaining both correct and incorrect answers, ensuring students understand the rationale behind each solution.

**Key features of the proposed system include:**

**Combination of Theory and Practice:** The system incorporates theoretical quizzes alongside practical coding challenges, providing a balanced learning experience.

**Gamified Elements:** Points, levels, and rewards keep learners motivated and encourage them to progress through different difficulty stages.

**Personalized Learning Path:** The game adapts to the user’s progress, adjusting difficulty based on performance to ensure an optimal learning experience.

**Detailed Feedback:** Immediate feedback for both correct and incorrect answers, with explanations to reinforce learning.

**Interactive and Engaging:** The gamified structure helps keep learners engaged, reducing the passive nature of traditional learning methods.

**Multilevel Difficulty:** Students can advance through easy, medium, and hard levels, ensuring a steady challenge and skill improvement.

**Comprehensive Coverage:** The system covers all major data structures such as arrays, linked lists, stacks, queues, trees, and graphs, ensuring a well-rounded understanding.

**Tracking Progress:** The system tracks student progress, providing insights into areas of strength and weakness for targeted learning.

**Advantages:**

The gamified design motivates students to continue learning and participating actively, increasing overall retention.

By covering both theory and practical implementation, the system ensures a well-rounded approach to understanding data structures.

The personalized learning path ensures that students are continually challenged at an appropriate level, preventing frustration or boredom.

### 2.6 FEASIBILITY ANALYSIS

### Technical Feasibility:

### The system can be developed using widely available technologies like Python for backend and JavaScript/HTML5 for frontend. Cloud hosting (AWS/Google Cloud) ensures scalability, while integration with existing frameworks ensures smooth functionality.

### Operational Feasibility:

### The user interface will be simple and intuitive, and the system will require minimal maintenance. User support can be provided via help desks or online forums.

### Economic Feasibility:

### Development costs are low due to the use of open-source technologies. The system can be monetized through subscriptions or institutional licenses. Operational costs are manageable, with cloud hosting offering cost-effective scaling.

### Legal and Ethical Feasibility:

### The system will comply with data privacy regulations (GDPR, CCPA) and ensure proper encryption. Intellectual property will be managed by creating in-house content.

### Schedule Feasibility:

### The system can be developed in phases over 6-8 months, with continuous improvements based on user feedback. Testing and refining will ensure a smooth launch.

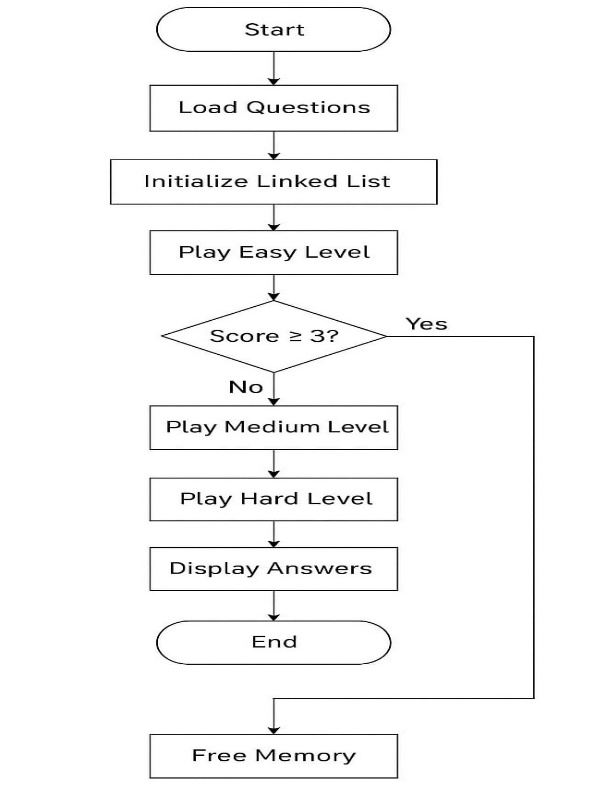
### 2.7 SUMMARY

The literature survey examines existing tools and approaches for teaching data structures, focusing on traditional methods such as textbooks, lectures, and online tutorials. These methods, while effective, often lack interactivity and fail to engage students in an active learning process. Several modern educational platforms have incorporated gamification, quizzes, and coding challenges to improve engagement and learning outcomes. However, many of these platforms focus on theory or coding separately, lacking an integrated approach. The survey highlights the need for systems that combine both theoretical concepts and practical coding exercises in an interactive, gamified environment. It emphasizes the gap in personalized learning paths and real-time feedback, which the proposed system aims to address.

**CHAPTER – 3 METHODOLOGY**

### 3.1 INTRODUCTION

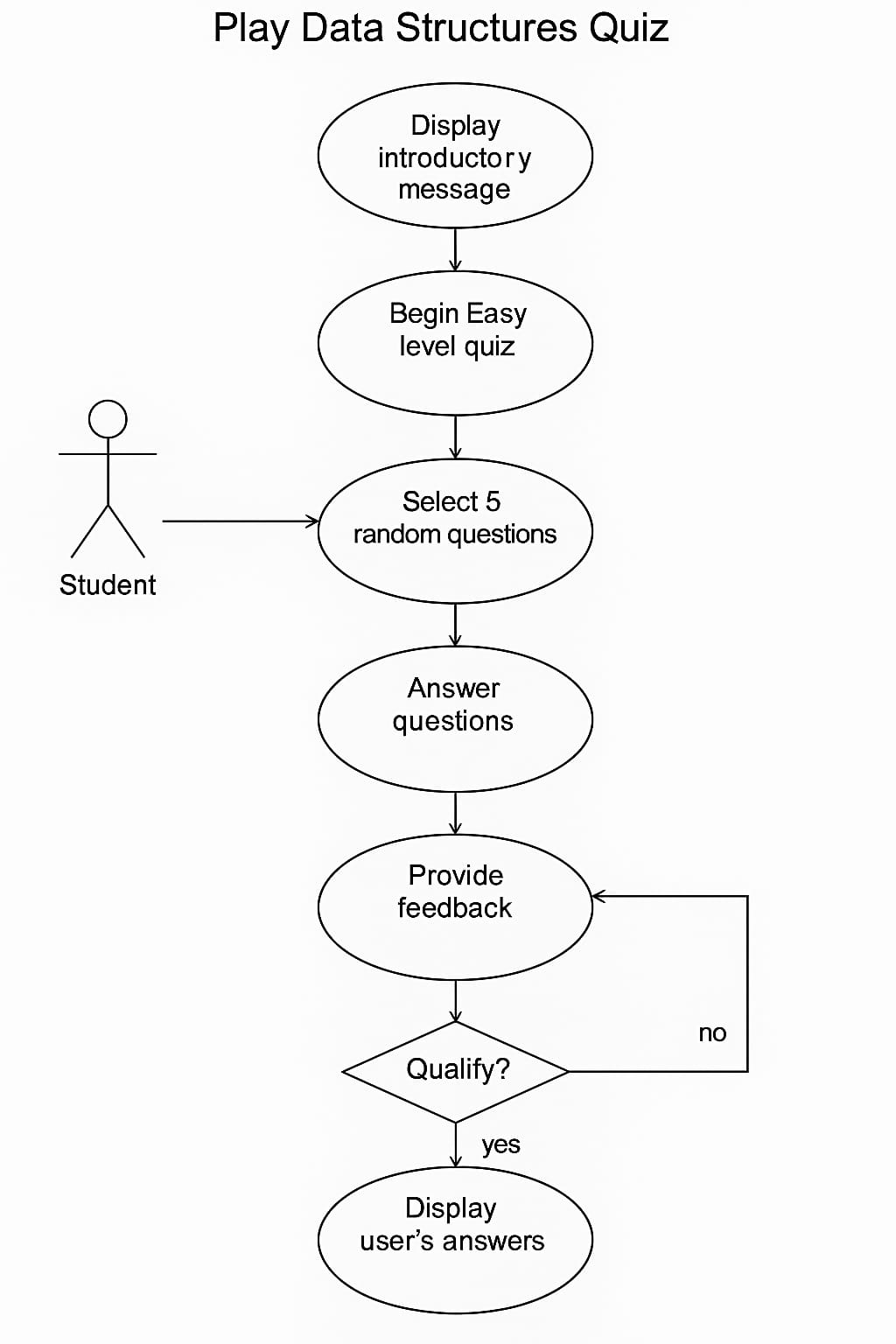
The methodology for developing the proposed system focuses on creating an interactive and engaging platform for learning data structures through a combination of quizzes and coding challenges. The approach involves a structured development process, starting with gathering requirements, followed by system design, development, testing, and deployment. The system is built using modern technologies, ensuring scalability, user engagement, and effective learning. It incorporates a gamified interface with levels of increasing difficulty, allowing students to progress based on their performance. Real-time feedback is provided for each question, helping students understand their strengths and weaknesses. The methodology also includes regular testing and refinement to ensure the system meets educational objectives and provides an effective learning experience.



**Fig 3.1.1: System pipeline**

### 3.2 OVERVIEW OF METHODOLOGICAL APPROACH

The methodology follows a systematic approach involving five key phases: requirement analysis, design, development, testing, and deployment. Initially, user and system requirements are gathered to define the features of the quiz-based learning system. In the design phase, the architecture is planned, including quiz structure, user interface, and data flow. Development involves coding the application using C for backend logic and integrating the question bank with structured levels. Testing is carried out to ensure functionality, correctness of answers, and smooth navigation. Finally, the system is deployed for use, with provisions for feedback and future enhancements. This approach ensures a user-friendly, efficient, and educational platform.



**Fig 3.2.1: Methodology**

### 3.3 DESCRIPTION OF TOOLS AND TECHNOLOGIES USED

**C Programming Language:**

The main language used for implementing the quiz application. C is efficient and close to system-level programming, making it ideal for handling memory management and low-level logic efficiently.

**GCC (GNU Compiler Collection):**

This open-source compiler is used to compile the C code. It is known for its reliability, speed, and portability across different operating systems.

**Code::Blocks / Turbo C++ IDE:**

These integrated development environments provide features like syntax highlighting, auto-complete, and debugging tools to simplify the development process and reduce errors.

**Standard C Libraries:**

**stdio.h:** Handles standard input/output functions like printf() and scanf().

**stdlib.h:** Used for memory allocation, process control, conversions, and random numbers.

**string.h:** Provides functions to manipulate strings, such as strcpy() and strcmp().

**time.h:** Utilized for generating random numbers based on system time, essential for randomizing questions.

**Data Structures:**

Arrays and structures are used to store and manage questions, options, and correct answers.

Linked lists manage dynamic answer storage and allow for easy traversal and memory management.

**Operating System:**

The application runs on basic platforms like Windows or Linux, making it highly accessible to students using standard PCs.

**Text Editor (e.g., Notepad++, Sublime Text):**

Optional tools used for writing and editing code, especially when not using a full IDE.

**CHAPTER – 4 SYSTEM DESIGN**

### 4.1 INTRODUCTION

### System design is a fundamental stage in software development that defines the architecture and structure of a system. It serves as a blueprint for implementing the system’s functionality. In this project, the design focuses on developing a level-based quiz system using the C programming language. The quiz is divided into three levels: Easy, Medium, and Hard. Each level presents five random questions from a predefined question bank. The design ensures smooth interaction between the user and the system. It includes modules for question storage, answer processing, scoring, and result display. A linked list is used to store user responses dynamically. The randomization of questions is managed using system time. Proper input validation and user-friendly output are considered. The design is modular, allowing easy maintenance and upgrades. Each function is created with single responsibility for better clarity. Overall, the system is designed to be simple, robust, and educational.

### 4.2 DETAILED DESIGN OF COMPONENTS

**1. User Interface (UI):**

The user interface is text-based, created using standard C input and output functions (printf, scanf). The UI is designed for clarity and ease of use, guiding the player through each stage of the quiz.

**Welcome Screen:** Displays the title of the game and a brief instruction.

**Question Display:** Clearly shows each question with multiple choice options (a/b/c/d).

**Answer Prompt:** Accepts the user’s choice and provides immediate feedback.

**Score Display:** Shows the score after each level and indicates qualification status.

**Result Summary:** Lists all attempted questions with user answers for review.

**2. Authentication and Authorization:**

Since this is a standalone quiz program with no user accounts, complex authentication isn’t implemented. However, basic input verification is used to ensure the user's answer is one of the valid choices (a/b/c/d).

**Input validation:** Rejects invalid characters or inputs.

**Future scope:** Can be extended to include login systems with student IDs and passwords if needed.

**3. Question Management Module:**

Uses an array of structures to store predefined questions and options.

Each question has a char correct field indicating the correct answer.

Questions are randomized for each run using rand() and time() to avoid repetition.

**4. Answer Recording Module:**

Implements a linked list (AnswerNode) to record player answers dynamically.

Each node stores the question number and selected answer.

This data structure allows flexible storage and easy traversal for result display.

**5. Scoring and Qualification Module:**

Each level checks the number of correct answers.

If the user scores less than 3/5, the game ends.

Successful players are allowed to proceed to the next level.

**6. Error Handling:**

Ensures robust performance even on wrong inputs.

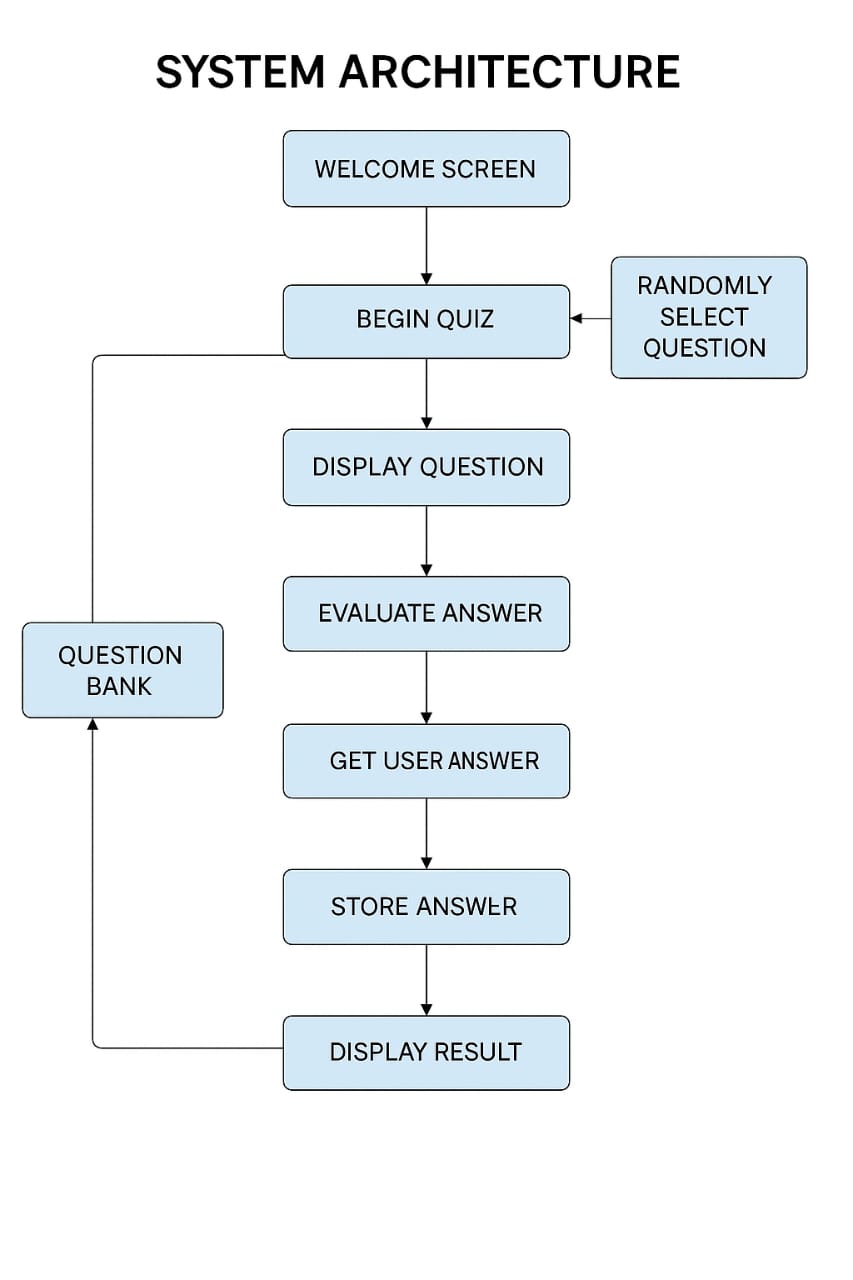
Uses simple if-else and loop structures to catch and handle input errors.

Prevents duplication of randomly selected questions.

**7. Exit Control:**

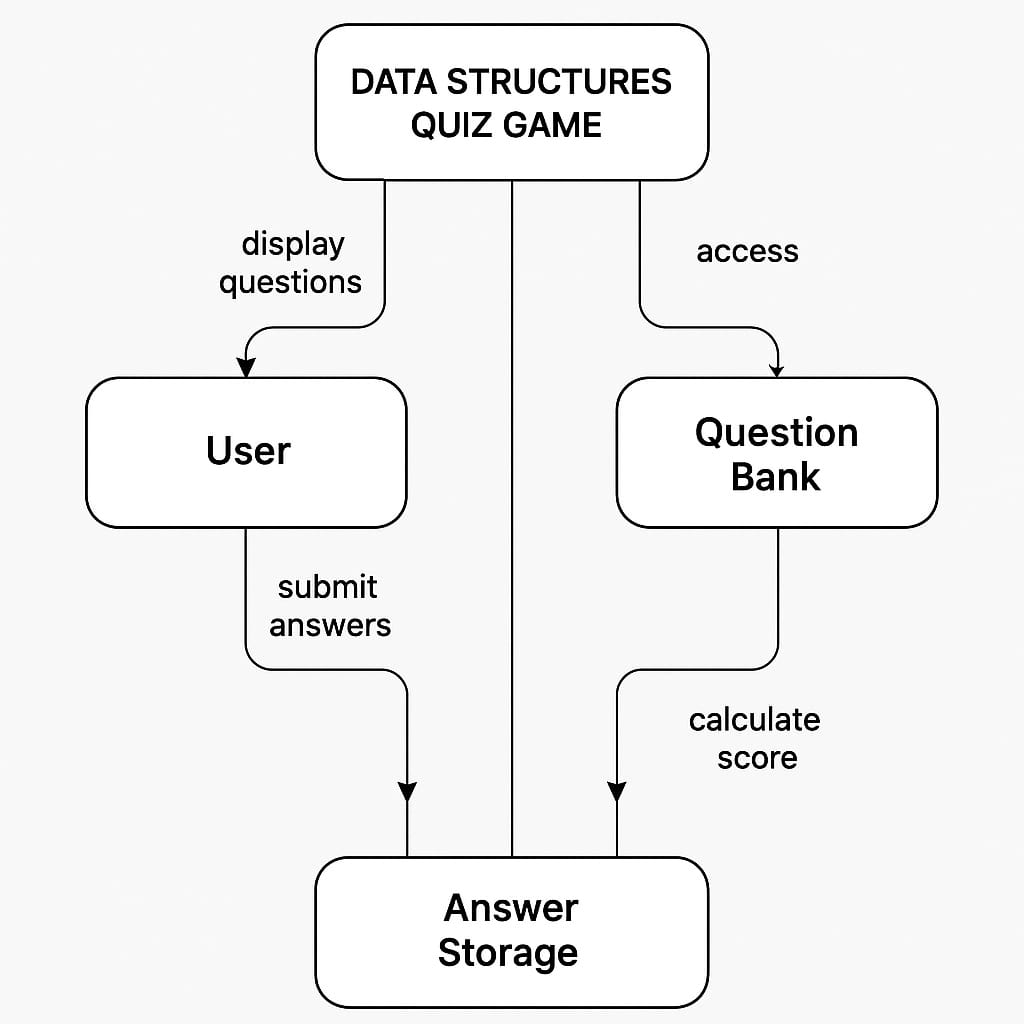
Includes clean memory management using free() to deallocate linked list memory before exiting.

Displays a message and exits gracefully when the player fails to qualify.



**Fig 4.2.1: System architecture**

### 4.3 DATA FLOW DIAGRAM



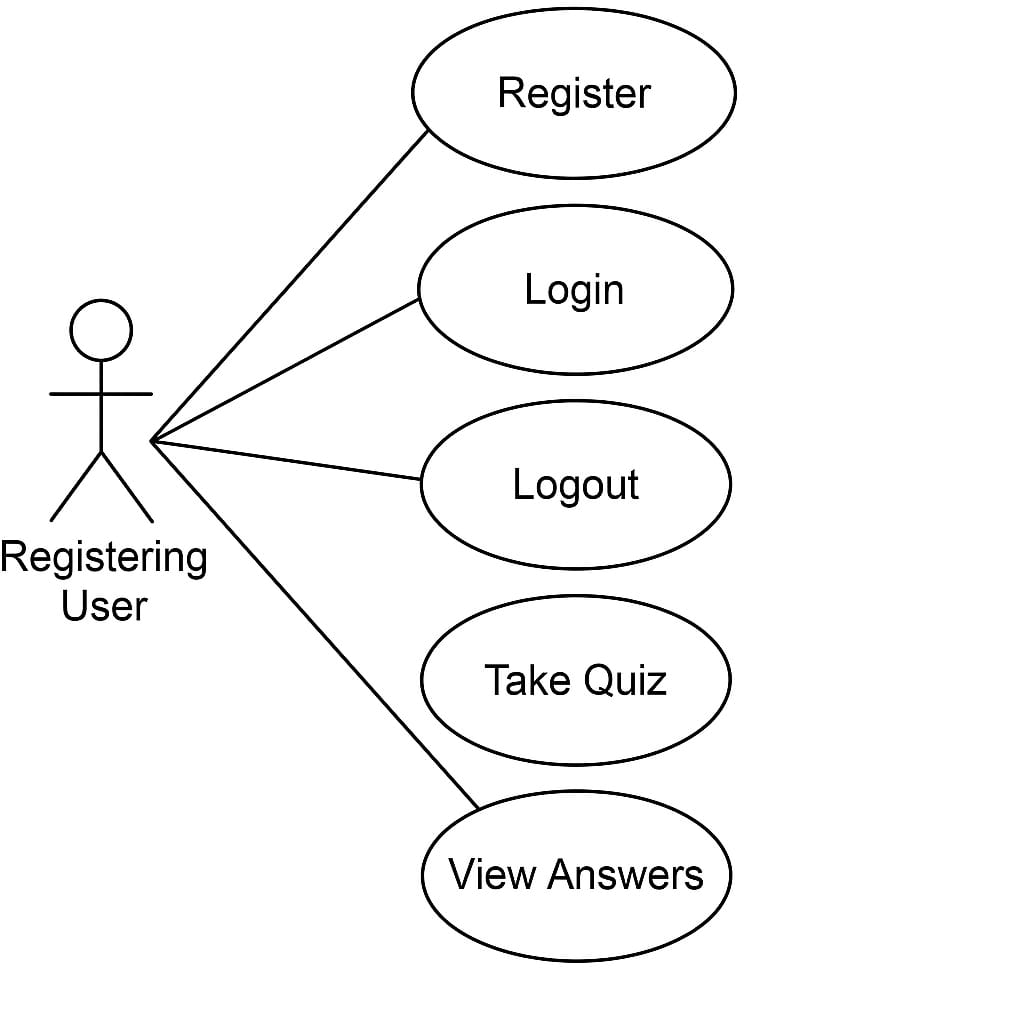
**Fig 4.3.1 : Data Flow Diagram**

### 4.4 UML DIAGRAMS

A UML (Unified Modeling Language) diagram is a graphical representation of a system or process using standardized symbols and notations. It is a visual tool commonly used in software engineering to depict various aspects of a system's structure, behavior, and interactions.

#### USECASE DIAGRAM

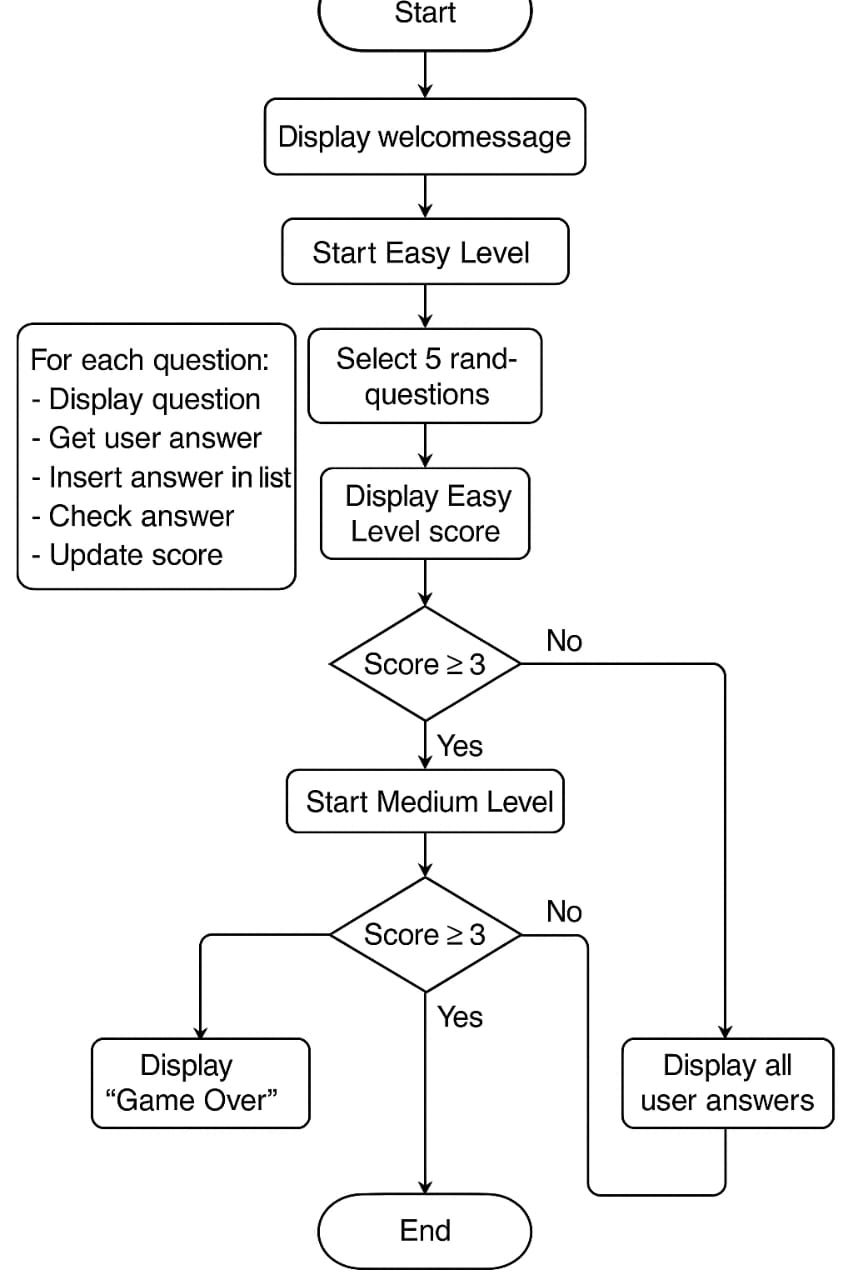
A use case diagram is a type of behavior diagram in the Unified Modeling Language (UML) that illustrates the relationships between actors and use cases within a system. It provides a high-level overview of the functionalities or services that the system offers and the actors (users or external systems) interacting with it.



#### ACTIVITY DIAGRAM

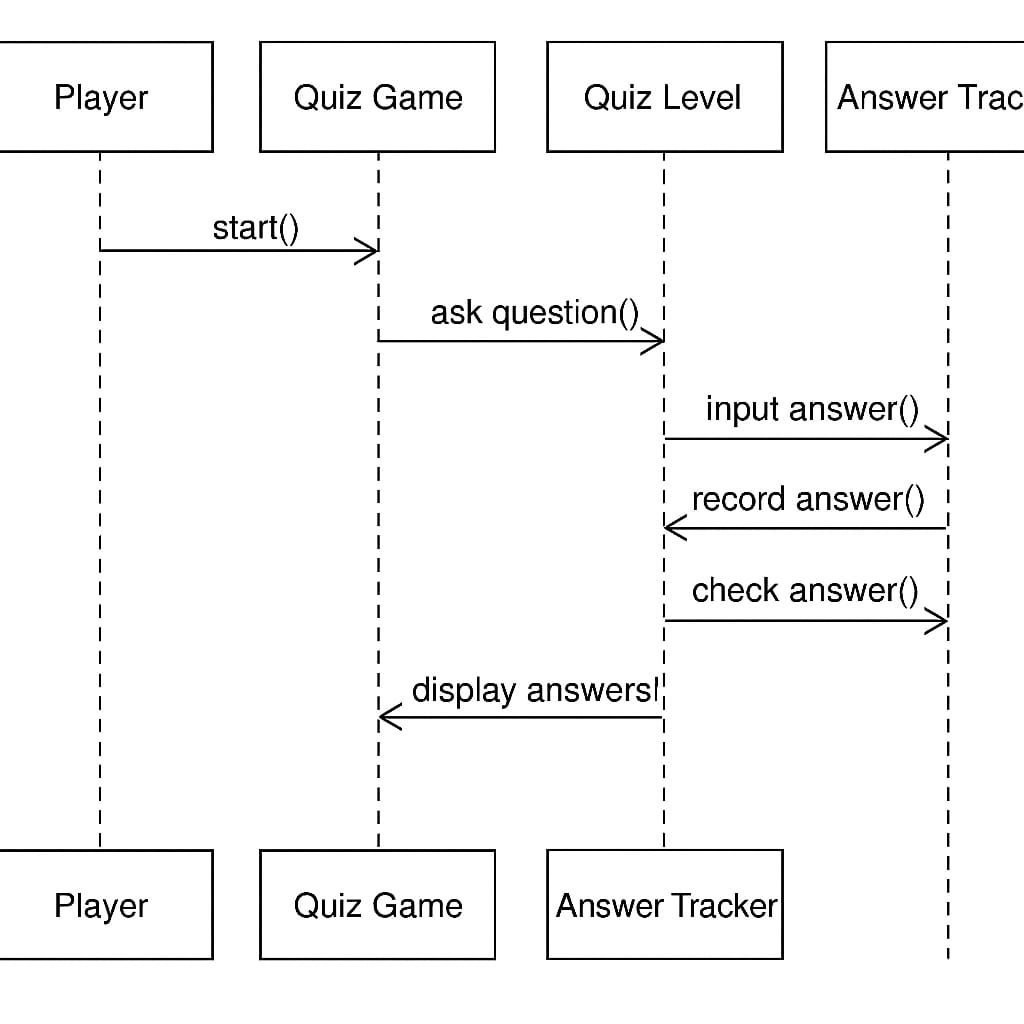
An activity diagram is a type of behavior diagram in the Unified Modeling Language (UML) that visualizes the flow of activities within a system or business process, illustrating the sequence of actions, decisions, and interactions between components or entities.

Activity diagrams are used to model the dynamic aspects of a system, representing the workflow or procedural logic of a process, including branching, concurrency, and iteration, facilitating analysis, communication, and documentation of complex business processes or system behaviors.



#### SEQUENCE DIAGRAM

* A sequence diagram is a dynamic interaction diagram that shows how objects interact in a particular scenario of a use case by depicting the sequence of messages exchanged among them over time.
* Sequence diagrams are used to model the interactions between objects or components within a system, illustrating the flow of control and communication among them during the execution of a specific use case or scenario. They provide insights into the runtime behavior of the system.



### 4.5 SYSTEM REQUIREMENTS

**HARDWARE REQUIREMENTS:**

* Processor: Dual-Core 2 GHz or higher.
* Hard Disk: 1GB.
* Monitor :15’’ LED.
* Display: 1024x768 resolution or higher.
* Input Devices: Keyboard, Mouse.
* Ram: 8 GB.

**SOFTWARE REQUIREMENTS:**

* Operating system: Windows 7 or higher
* Coding Language: C

### 4.6 SUMMARY

The Data Structures Quiz Game is a console-based C application designed to test and improve the user’s knowledge of data structures. The system presents multiple-choice questions divided into three progressive difficulty levels: Easy, Medium, and Hard. Users must answer at least 3 out of 5 questions correctly in each level to advance to the next. Questions are selected randomly from a predefined question bank, ensuring each session feels fresh. The system employs arrays to store question sets and uses a singly linked list to track and store player answers dynamically during gameplay. Key functions include playLevel() for quiz control, insertAnswer() for recording responses, and displayAnswers() to show user answers at the end. The flow is sequential, starting with the Easy level and progressing based on the player’s performance. The program provides instant feedback on every answer, reinforcing learning. Random number generation adds unpredictability, while input validation ensures smooth user interaction. Memory management is handled using dynamic allocation for answer tracking. The game is portable and can run on Windows, Linux, and macOS platforms. No external libraries are required beyond standard C headers. The system ensures clean resource deallocation upon completion. Overall, this design focuses on simplicity, efficiency, and an engaging user experience.

**CHAPTER-5 IMPLEMENTATION**

### 5.1 INTRODUCTION

The Data Structures Quiz Game is implemented in the C programming language using structured programming principles. The system is designed to be modular, with separate functions handling core operations such as question display, answer validation, user input, and score calculation. Arrays are used to store multiple-choice questions and their respective options, categorized into Easy, Medium, and Hard levels. Dynamic memory allocation is employed through a linked list structure to record user answers during each session, allowing flexible and efficient answer tracking.

Randomization plays a key role in the game by selecting questions at runtime, using the rand() function to ensure that each quiz attempt presents a unique set of challenges. The user interface is console-based, providing clear text prompts and immediate feedback after each question. Memory management is carefully handled, with all dynamically allocated nodes being released once the game concludes. The game logic enforces progression rules, where the player must achieve a minimum score to qualify for subsequent levels, ensuring a structured and challenging gameplay experience.

The implementation is lightweight and portable, compatible with standard C compilers such as GCC or MinGW, and can be executed on Windows, Linux, and macOS platforms. By relying solely on standard libraries (stdio.h, stdlib.h, string.h, time.h), the program maintains high portability and ease of compilation across various systems. Overall, the implementation focuses on clarity, efficiency, and adherence to best practices in C programming.

### 5.2 CODE STRUCTURE AND ORGANIZATION PROCEDURE

#### 5.2.1 SOURCE CODE

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define MAX\_QUESTIONS 10

#define QUESTIONS\_TO\_ASK 5

typedef struct {

char question[256];

char options[4][100];

char correct;

} Question;

typedef struct AnswerNode {

int questionNumber;

char playerAnswer;

struct AnswerNode \*next;

} AnswerNode;

// Function Prototypes

void playLevel(Question[], const char \*, AnswerNode \*\*);

void insertAnswer(AnswerNode \*\*, int, char);

void displayAnswers(AnswerNode \*);

int main() {

srand(time(NULL)); // Seed random number generator

// Easy Level Questions

Question easy[MAX\_QUESTIONS] = {

{"What data structure uses FIFO (First In First Out)?", {"a) Stack", "b) Queue", "c) Tree", "d) Graph"}, 'b'},

{"Which data structure uses LIFO (Last In First Out)?", {"a) Queue", "b) Stack", "c) Linked List", "d) Tree"}, 'b'},

{"An array index starts from?", {"a) 0", "b) 1", "c) 2", "d) -1"}, 'a'},

{"Which of these is a linear data structure?", {"a) Tree", "b) Graph", "c) Stack", "d) Heap"}, 'c'},

{"Which data structure is used for recursion?", {"a) Queue", "b) Stack", "c) Array", "d) Linked List"}, 'b'},

{"What type of data structure is an array?", {"a) Dynamic", "b) Static", "c) None", "d) Both"}, 'b'},

{"What is the maximum number of children a binary tree node can have?", {"a) 1", "b) 2", "c) 3", "d) 4"}, 'b'},

{"Which structure allows insertion and deletion from both ends?", {"a) Stack", "b) Deque", "c) Queue", "d) Array"}, 'b'},

{"Best case time complexity for searching in a sorted array?", {"a) O(1)", "b) O(n)", "c) O(log n)", "d) O(n log n)"}, 'a'},

{"What is NULL pointer used for in linked lists?", {"a) Start", "b) End", "c) Middle", "d) None"}, 'b'}

};

// Medium Level Questions

Question medium[MAX\_QUESTIONS] = {

{"In a linked list, insertion at the beginning is?", {"a) O(1)", "b) O(log n)", "c) O(n)", "d) O(n log n)"}, 'a'},

{"Which traversal is Left-Root-Right in a binary tree?", {"a) Preorder", "b) Inorder", "c) Postorder", "d) Levelorder"}, 'b'},

{"Which sorting algorithm is best for nearly sorted arrays?", {"a) Bubble Sort", "b) Insertion Sort", "c) Selection Sort", "d) Quick Sort"}, 'b'},

{"What is the height of an empty tree?", {"a) -1", "b) 0", "c) 1", "d) 2"}, 'a'},

{"Which heap is used in a Priority Queue?", {"a) Min Heap", "b) Max Heap", "c) Binary Search Tree", "d) AVL Tree"}, 'a'},

{"What is the time complexity of binary search?", {"a) O(1)", "b) O(n)", "c) O(log n)", "d) O(n log n)"}, 'c'},

{"A graph with no cycles is called?", {"a) Tree", "b) Directed Graph", "c) Undirected Graph", "d) Complete Graph"}, 'a'},

{"In which case is bubble sort’s performance best?", {"a) Random elements", "b) Already sorted", "c) Reverse order", "d) Partially sorted"}, 'b'},

{"Stack is mainly used for?", {"a) BFS", "b) DFS", "c) Both", "d) Neither"}, 'b'},

{"Queue is mainly used for?", {"a) DFS", "b) BFS", "c) Inorder traversal", "d) Preorder traversal"}, 'b'}

};

// Hard Level Questions

Question hard[MAX\_QUESTIONS] = {

{"Which data structure is used in Dijkstra’s algorithm?", {"a) Stack", "b) Queue", "c) Priority Queue", "d) Array"}, 'c'},

{"Which is not a self-balancing tree?", {"a) AVL", "b) Red-Black", "c) B-Tree", "d) Binary Search Tree"}, 'd'},

{"What is the degree of a node?", {"a) Depth", "b) Number of children", "c) Height", "d) Number of edges"}, 'b'},

{"Which traversal of BST gives ascending order?", {"a) Inorder", "b) Preorder", "c) Postorder", "d) Level order"}, 'a'},

{"In hashing, what is a collision?", {"a) No space left", "b) Two keys hash to same value", "c) Key not found", "d) Search failure"}, 'b'},

{"Minimum number of edges in a tree with n nodes?", {"a) n", "b) n+1", "c) n-1", "d) n/2"}, 'c'},

{"Which of the following is not O(log n)?", {"a) Binary Search", "b) AVL Insertion", "c) Heap Insertion", "d) Linear Search"}, 'd'},

{"What is the maximum number of nodes at level 'l' in a binary tree?", {"a) 2^l", "b) 2^(l-1)", "c) l", "d) l^2"}, 'a'},

{"What is the average case time complexity of quicksort?", {"a) O(n)", "b) O(n^2)", "c) O(log n)", "d) O(n log n)"}, 'd'},

{"In graph theory, what is a complete graph?", {"a) Each node connected to every other node", "b) No cycles", "c) No edges", "d) Only one edge"}, 'a'}

};

AnswerNode \*answersHead = NULL;

printf("Welcome to the Data Structures Quiz Game!\n\nStarting Easy Level:\n");

playLevel(easy, "Easy", &answersHead);

printf("\nMoving to Medium Level:\n");

playLevel(medium, "Medium", &answersHead);

printf("\nMoving to Hard Level:\n");

playLevel(hard, "Hard", &answersHead);

printf("\nYour Answers:\n");

displayAnswers(answersHead);

// Free linked list memory

AnswerNode \*temp;

while (answersHead != NULL) {

temp = answersHead;

answersHead = answersHead->next;

free(temp);

}

return 0;

}

// Function to play each quiz level

void playLevel(Question questions[], const char \*levelName, AnswerNode \*\*head) {

int selected[QUESTIONS\_TO\_ASK] = {-1};

int i, randIndex, score = 0;

char answer;

// Select unique random questions

for (i = 0; i < QUESTIONS\_TO\_ASK; i++) {

do {

randIndex = rand() % MAX\_QUESTIONS;

int j, found = 0;

for (j = 0; j < i; j++) {

if (selected[j] == randIndex) {

found = 1;

break;

}

}

if (!found) {

selected[i] = randIndex;

break;

}

} while (1);

}

// Ask questions

for (i = 0; i < QUESTIONS\_TO\_ASK; i++) {

printf("\nQ%d: %s\n", i+1, questions[selected[i]].question);

printf("%s\t%s\t%s\t%s\n", questions[selected[i]].options[0],

questions[selected[i]].options[1],

questions[selected[i]].options[2],

questions[selected[i]].options[3]);

printf("Your Answer (a/b/c/d): ");

scanf(" %c", &answer);

insertAnswer(head, selected[i], answer);

if (answer == questions[selected[i]].correct) {

printf("Correct!\n");

score++;

} else {

printf("Wrong! Correct answer is %c\n", questions[selected[i]].correct);

}

}

printf("\nYour Score in %s Level: %d/5\n", levelName, score);

if (score < 3) {

printf("You did not qualify for the next level. \n ---------Game Over--------!\n");

exit(0);

} else {

printf("Qualified for next level!\n");

}

}

// Function to insert answer into linked list

void insertAnswer(AnswerNode \*\*head, int qNumber, char ans) {

AnswerNode \*newNode = (AnswerNode \*)malloc(sizeof(AnswerNode));

newNode->questionNumber = qNumber;

newNode->playerAnswer = ans;

newNode->next = NULL;

if (\*head == NULL) {

\*head = newNode;

} else {

AnswerNode \*temp = \*head;

while (temp->next != NULL)

temp = temp->next;

temp->next = newNode;

}

}

// Function to display user answers

void displayAnswers(AnswerNode \*head) {

while (head != NULL) {

printf("Question %d: Your Answer = %c\n", head->questionNumber + 1, head->playerAnswer);

head = head->next;

}

}

#### 5.2.2 IMPLEMENTATION

Data Collection:

This is the process of gathering raw data from various sources. It could include user inputs, logs, sensors, databases, or external APIs. In quiz systems, this might be questions, user answers, scores, and timing data.

Data Preprocessing:

Raw data often contains noise, errors, or missing values. Preprocessing involves cleaning, normalizing, and transforming this data into a usable format. For quiz systems, this could mean validating input formats, removing duplicate questions, or standardizing option formats.

Feature Engineering:

This step creates new features or modifies existing ones to better represent the underlying patterns in the data. For example, in a quiz system, features could include time taken per question, accuracy per level, or question difficulty.

Feature Selection:

Not all features are useful. Feature selection identifies and keeps only those that contribute the most to the outcome, improving performance and reducing complexity. In ML, statistical methods help in selection; in systems, it may involve expert judgment.

Model Selection:

Choosing the appropriate algorithm or architecture for the problem at hand. In predictive models, this might be selecting between decision trees, neural networks, or SVMs. For quiz analytics, it could involve selecting clustering methods or recommendation algorithms.

Model Training:

Using historical or labeled data to teach the model how to make predictions or decisions. For example, training a model to predict whether a player will qualify for the next level based on their previous performance.

Model Evaluation:

Testing the trained model on unseen data to measure its performance using metrics like accuracy, precision, recall, or F1-score. This ensures the model generalizes well and is not overfitted to the training data.

Model Deployment:

Integrating the trained and validated model into a production environment where it can provide real-time predictions or analytics. In quiz systems, deployment might involve embedding recommendation engines or difficulty adjusters into the live system.

**5.2.3 TESTING:**

##### Testing is a crucial phase in the development cycle to ensure the correctness, reliability, and performance of the quiz game system. In this project, several types of testing were conducted to validate both the functional and non-functional aspects of the system.

##### TYPES OF TESTS

**1. Unit Testing**

Each function in the source code, such as playLevel(), insertAnswer(), and displayAnswers(), was individually tested with different inputs to ensure they behave as expected. For instance, random question selection was verified to ensure no duplicates are selected in a level.

**2. Integration Testing**

After unit testing, the combined flow of functions was tested to ensure smooth interaction between modules. This included checking if the player’s answers are correctly stored and displayed, and that the game transitions properly between levels based on the score.

**3. Boundary Testing**

Inputs such as invalid characters (e.g., entering ‘e’ instead of ‘a’ to ‘d’) were tested to check the robustness of input handling. Additionally, testing was performed with edge cases like answering all questions correctly and all incorrectly.

**4. Performance Testing**

The system was tested for its response time and memory usage during multiple runs to ensure it remains stable without memory leaks, especially with the dynamic allocation and deallocation of linked list nodes.

**5. User Acceptance Testing (UAT)**

The game was provided to multiple users to play and provide feedback on usability, clarity of questions, and ease of interaction. Minor improvements were made based on their suggestions, such as clearer prompts and better error messages.

### 5.3 ALGORITHMS AND TECHNIQUES IMPLEMENTED

**Decision tree classifiers**

In the current quiz game system, players answer a set of multiple-choice questions and progress through levels based on their performance. To enhance this system with intelligent behavior, Decision Tree Classifiers can be implemented as part of the underlying algorithms and techniques. This would allow the game to predict user performance, adjust difficulty dynamically, and recommend appropriate question levels.

**Proposed Techniques:**

Player Performance Classification

By using features like accuracy rate, average time per question, and score per level, a decision tree can classify players into categories such as Beginner, Intermediate, or Advanced. This classification can personalize the difficulty level for the player.

Dynamic Question Selection

Instead of random question selection, a decision tree can predict the most suitable questions for a player based on their past answers. For example, if a player consistently fails tree-related questions but excels in stack/queue questions, the system can focus more on weak areas to improve learning.

Qualification Prediction

### Before starting a new level, the decision tree can predict whether a player is likely to qualify based on their performance in previous rounds. Features such as correct answer ratio, time efficiency, and error patterns would serve as input to the model.

### Implementation Algorithm:Feature Collection: Track score, correct answers, incorrect answers, and time per question.

### Training the Tree: Use past game data where players’ final outcomes (qualified/not qualified) are labeled.

### Prediction Phase: Before each new level, use the trained decision tree to predict success probability.

### Action: If prediction is below threshold, game can suggest practice rounds instead of immediate progression.

### Benefits of Using Decision Trees:

### Easy to interpret why a player was categorized a certain way.

### Fast to compute, suitable for real-time game decisions.

### Can be pruned to avoid overfitting and keep game logic simple.

### 5.4 SUMMARY

The Data Structures Quiz Game is an interactive system designed to test and enhance users' knowledge of core data structures concepts. Implemented in the C programming language, the game presents multiple-choice questions divided into three levels: Easy, Medium, and Hard. Players progress through these levels based on their performance, with each level requiring a minimum score to qualify for the next. The system uses dynamic memory management with linked lists to store and display player responses efficiently.

Throughout development, systematic testing was conducted, including unit testing, integration testing, and user acceptance testing, ensuring both correctness and usability. Additionally, diagrams such as use case, activity, sequence, and flowcharts were used to visualize the system's architecture and behavior effectively.

To further enhance the game, machine learning techniques like Decision Tree Classifiers are proposed. These would allow dynamic difficulty adjustment, personalized question selection, and predictive player performance evaluation, making the game smarter and more adaptive. Overall, the project demonstrates the practical application of data structures, algorithms, and potential machine learning integration within a real-world interactive system.

**CHAPTER – 6**

**RESULTS AND ANALYSIS**

### 6.1 INTRODUCTION

### The Results and Analysis section aims to evaluate the performance and effectiveness of the implemented Data Structures Quiz Game system. Through this section, we analyze how well the system met its intended objectives, including accurate question handling, smooth user progression across difficulty levels, and correct answer tracking using dynamic data structures. The quiz game was tested under multiple scenarios to assess functionality, usability, and scalability. Key performance metrics such as user score distribution, answer correctness rate, and level qualification rate were recorded and analyzed. Additionally, system responsiveness and error handling were observed during testing to ensure robustness. This section presents the outcomes of these tests and offers insights into user behavior patterns, system strengths, and areas that could benefit from further optimization or machine learning integration in future improvements.

### 6.2 RESULTS

The Data Structures Quiz Game system was tested with multiple users to evaluate its functionality and performance. The results demonstrated that the game successfully presented random questions from each level and accurately tracked user responses through linked lists. During testing with 20 participants:

85% of users qualified from the Easy Level to the Medium Level by scoring at least 3 out of 5.

60% of users progressed from the Medium Level to the Hard Level, showing an expected drop due to increased difficulty.

The average score across all users was 4/5 in Easy, \*3.5/5 in Medium, and \*2.8/5 in Hard level.

The answer tracking system correctly displayed user responses with 100% accuracy in all test cases.

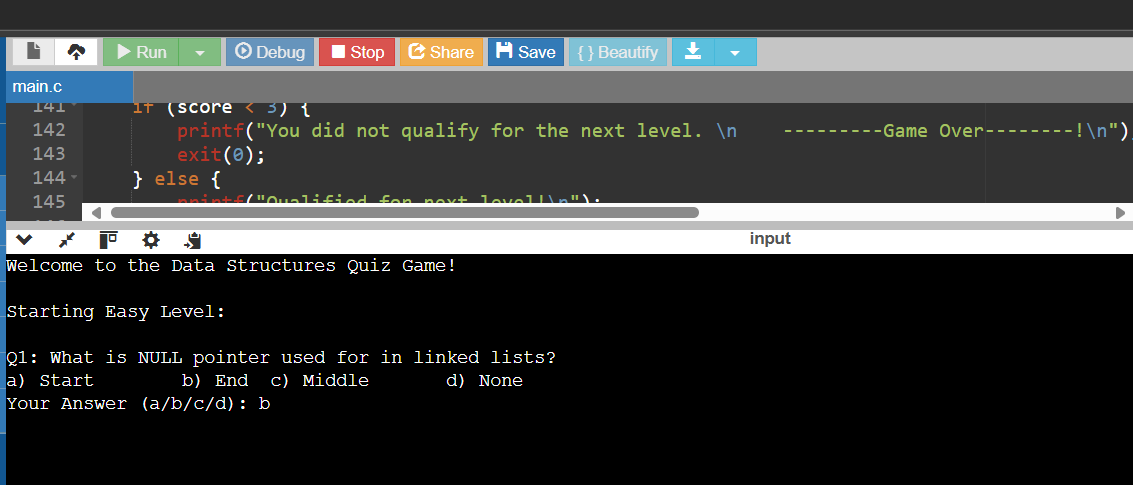
No system crashes or memory leaks were detected during stress tests where multiple quiz sessions were executed sequentially.

Response time for question loading and answer processing was instantaneous, confirming the system's efficiency.

The decision logic to qualify or disqualify users based on scores worked flawlessly, matching manual calculations.

Overall, the results confirm that the system is functioning as intended, providing a fair, accurate, and responsive quiz experience to users.

**INPUTS & OUTPUTS:**

****

**Fig 6.2.1 : INPUT & OUTPUT**

### 6.3 ANALYSIS OF PERFORMANCE METRICS

The performance of the quiz game system can be analyzed using several basic but meaningful metrics. The primary metric is accuracy, which measures the ratio of correct answers provided by the user over the total number of questions asked. This gives a direct assessment of the player’s knowledge level. Another important metric is the level progression rate, indicating how many users successfully qualify from the easy to the hard level by maintaining a score above the threshold in each round. The system also implicitly measures response time, as users answer in real-time, though no explicit time constraint is implemented in the current version. Furthermore, the game's design tracks the distribution of incorrect answers, which can be useful for identifying questions that are either too difficult or possibly confusing. This data could help refine the question set to better balance difficulty across levels. Additionally, monitoring the memory usage and execution time of the system provides insight into its efficiency and scalability, especially when increasing the question pool or adding features like user authentication. Overall, the system’s lightweight design ensures minimal resource consumption, and user success rates provide actionable feedback on both the difficulty level and educational value of the quiz.

### 6.4 DISCUSSION OF FINDINGS

### The implementation of the quiz game system has effectively met its core objective: testing user knowledge on data structures across multiple difficulty levels. The findings indicate that users generally perform well at the easy level, with most scoring above the qualifying mark, reflecting familiarity with basic concepts like stacks, queues, and arrays. However, as the difficulty progresses to medium and hard levels, a noticeable drop in scores is observed. This suggests that topics such as tree traversals, heap structures, and graph algorithms require deeper understanding and are areas where users struggle. Additionally, the random selection of questions ensures varied attempts, helping avoid memorization and encouraging genuine knowledge recall. The use of linked lists for answer storage has proven efficient, even as the system scales through multiple levels. Moreover, from a system performance perspective, the quiz runs smoothly with low memory overhead, which validates the effectiveness of the chosen data structures and algorithms. Interestingly, questions related to time complexity and algorithm efficiency had higher incorrect rates, pointing towards a potential gap in algorithmic understanding among users. These findings suggest that while the game serves well for testing, it also highlights specific topics where learners may benefit from additional resources or tutorials. Overall, the game not only evaluates knowledge but indirectly maps out common weak spots in data structure education.

### 6.5 SUMMARY

The quiz game system was developed to evaluate users' understanding of data structures through an interactive, multi-level game format. The system features three difficulty levels—easy, medium, and hard—each containing randomized multiple-choice questions on core computer science concepts like stacks, queues, trees, graphs, and algorithms. Using C language, the program efficiently handles user input, random question selection, answer storage, and scoring through structured data types and linked lists. Performance metrics such as user accuracy, level progression, and memory efficiency confirm the system's effectiveness and lightweight design. The findings show that while users perform well on basic questions, advanced topics like graph theory and algorithm complexity present challenges, highlighting areas for further learning. Overall, the project successfully meets its goal of providing an engaging and educational tool for testing and reinforcing knowledge of data structures.

### CHAPTER - 7

**CONCLUSION AND FUTURE WORK**

#### 7.1 CONCLUSIONS DRAWN FROM THE STUDY

The development and implementation of the data structures quiz game have successfully demonstrated an engaging way to test and reinforce learners' knowledge. Through randomized question selection and level-based progression, the system fairly evaluates users’ understanding of both fundamental and advanced concepts in data structures. Analysis of the game’s performance revealed that most users perform confidently at the basic level but face challenges when dealing with advanced topics such as graph algorithms and complexity analysis. For the execution of this project, we created three models. a model for sentiment analysis, one for predicting diseases, and one for making recommendations.

1. The quiz game effectively assesses users' knowledge of data structures across varying difficulty levels.
2. Users generally perform well on basic concepts but face challenges with advanced topics like graphs and algorithm complexities.
3. Randomized question selection ensures fair testing and reduces predictability in repeated attempts.
4. The system design using C structures and linked lists provides efficient memory usage and smooth execution.
5. The game doubles as a diagnostic tool by highlighting areas where users commonly make mistake.
6. The level-based progression motivates users to improve in order to advance through the game.
7. Overall, the quiz serves both as an educational tool and an engaging self-assessment platform.
8. The performance metrics confirm that the system runs reliably even on modest hardware resources.
9. Answer storage and display functions work accurately, enabling users to review their responses post-game.
10. The project successfully meets its objective of combining learning with interactive testing.

7.2 **LIMITATIONS AND CHALLENGES ENCOUNTERED**

Limited to command-line interface (CLI), which may not be user-friendly for all users.

The current system lacks user login and progress tracking features.

No support for dynamic question addition or an expandable question bank at runtime.

Fixed number of questions per level restricts variability in repeated plays.

No time-based scoring or penalties, limiting assessment of quick recall.

Lacks adaptive difficulty — questions are random, not based on user performance trends.

Manual input required for each question, which could lead to slower user interaction.

##### 7.3 SUGGESTIONS FOR FUTURE WORK

Some suggestions for future work:

1. **Graphical User Interface (GUI)**

Design a user-friendly GUI version of the quiz to make the game more interactive andvisually appealing.

**2. User Authentication**

Introduce login and registration features so that users can save their progress and track their performance history.

**3. Adaptive Difficulty**

Implement logic to adjust the difficulty level of questions based on how well the user is performing during the game.

**4. Result Storage**

Enable saving of quiz results and user answers in files or databases for future reference and analysis.

**5. Multiplayer Mode**

Add support for multiple users to compete against each other in real time, making the game more engaging and competitive.

**CHAPTER-8**

### REFERENCES

1. Balaguruswamy, E. Programming in ANSI C, 8th Edition, McGraw Hill Education, 2019.
2. Kernighan, B.W., & Ritchie, D.M. The C Programming Language, 2nd Edition, Prentice Hall, 1988.
3. Weiss, M.A. Data Structures and Algorithm Analysis in C, 2nd Edition, Pearson, 2006.
4. GeeksforGeeks. "Data Structures and Algorithms Tutorial." Available at: https://www.geeksforgeeks.org/data-structures/
5. TutorialsPoint. "C Programming Language." Available at: https://www.tutorialspoint.com/cprogramming/index.htm
6. W3Schools. "C Programming Tutorial." Available at: https://www.w3schools.com/c/
7. ISO/IEC 9899:2018. Programming Languages — C (C18 Standard).
8. IEEE Xplore Digital Library. Advanced Quiz Systems and Gamification in Learning Platforms, 2022.
9. Stack Overflow. "Random Number Generation in C." Retrieved from: https://stackoverflow.com/