ICT Project 1 – Final Report

Interactive Quiz Application Using C++ and Embarcadero RAD Studio 13

Submitted by:

Dipu Mondol (ID: IT-24040)

Limon Hasan (ID: IT-24041)

Course: ICT Project 1 (Course Code: ICT 1200)

Adviser: Mr. Ziaur Rahman (Associate Professor)

Department of Information and Communication Technology

Date of Submission: 14 November 2025

Acknowledgment

We would like to express our sincere gratitude to **Mr. Ziaur Rahman**, Associate Professor, for his continuous guidance, constructive feedback, and encouragement throughout the completion of our ICT Project 1 course. His advice helped us refine our ideas and strengthen the technical and structural aspects of our project, *"Interactive Quiz Application Using C++ and Embarcadero RAD Studio 13."*

We would also like to thank the **Department of Information and Communication Technology** for providing access to resources, tools, and laboratory facilities that enabled us to test and evaluate our application. Finally, we thank our classmates and peers for their valuable input during the design and presentation phases of this project.

This project has been an invaluable learning experience, helping us to apply theoretical programming knowledge to a practical, real-world graphical user interface (GUI) system.

Dipu Mondol (IT-24040)

Limon Hasan (IT-24041)

Table of Contents

---

1. Introduction

This report presents the design, implementation, and testing of a GUI-based quiz application developed using **C++** in the **Embarcadero RAD Studio 13** environment. The project aims to demonstrate how graphical user interfaces can be integrated with object-oriented programming and data structures to create interactive educational software.

The primary purpose of this project is to design a program that allows users to participate in a short quiz consisting of multiple-choice questions. The system provides an interactive graphical interface, enabling users to select answers using radio buttons, confirm their responses, and track their scores dynamically as they progress through the quiz.

In this project, we have implemented fundamental programming concepts such as **classes**, **constructors**, **event handling**, and **queues** to manage a sequence of questions efficiently. Using Embarcadero's **FireMonkey (FMX)** framework, we built a visually appealing, platform-independent application that can run smoothly on Windows.

This project aligns with the course objectives of **ICT Project 1 (ICT 1200)**, which emphasizes hands-on software design, problem-solving, and user interface development. It represents a practical implementation of theoretical knowledge acquired from previous coursework, including programming fundamentals, object-oriented design, and human-computer interaction.

The key deliverables of this project include:

- A working quiz application built in RAD Studio using C++.
- A structured and modular source code implementation.
- A user-friendly GUI with labeled buttons, text fields, and score indicators.
- A demonstration of event-driven programming principles.
- A comprehensive project report documenting the design, development, and evaluation processes.

The following sections of this report discuss the client background, problem definition, design methodology, system architecture, implementation, and testing procedures, followed by a user manual for installation and operation.

---

2. Client / Instructor Background

The project was supervised by **Mr. Ziaur Rahman**, Associate Professor, Department of Information and Communication Technology.
Mr. Rahman specializes in programming methodologies, software engineering, and user interface design. Throughout the project, he provided detailed technical guidance, emphasizing

both the algorithmic structure and the user interface experience. His supervision ensured that the final application adhered to software design principles and academic integrity standards.

The guidance provided by Mr. Rahman reflects a commitment to practical learning — encouraging students to move beyond code syntax and focus on usability, error handling, and visual appeal. His mentoring style helped transform the project from a simple console-based quiz to a complete GUI-driven application.

---

3. The Problem

The goal of this project was to design and implement an **interactive quiz system** that enables users to answer multiple-choice questions within a clean and easy-to-navigate graphical interface. The need for this project arises from the limitations of traditional console-based quiz programs, which often lack visual clarity, intuitive controls, and engaging user feedback.

The problem can be summarized as follows:

1. **Limited Interactivity:** Console-based quizzes require textual input, which can be confusing for beginners or younger learners.
2. **Poor Usability:** Lack of GUI elements such as buttons, labels, and score counters reduces user engagement.
3. **Static Design:** Hard-coded questions are difficult to update or manage.
4. **Educational Gap:** Many simple quizzes fail to provide immediate feedback or score tracking during execution.

Our objective was to design a GUI application that resolves these limitations using C++'s object-oriented capabilities and the FMX library's graphical controls. The system had to:

- Display a question and three possible answers.
- Allow users to select one answer at a time.
- Verify the correctness of the selected answer.
- Keep track of the user's total score.
- Automatically load the next question after confirmation.
- Display a completion message when all questions are answered.

---

4. Design Process

4.1 Overall Design Process

The design process of the quiz application followed an **iterative development approach**, consisting of four primary stages:

1. **Requirement Analysis:** Understanding the functional and non-functional requirements of the quiz system.
2. **System Design:** Designing the class structure, GUI layout, and question-loading mechanism.
3. **Implementation:** Coding the application using C++ in the Embarcadero RAD Studio 13 environment.

4. **Testing and Evaluation:** Running test cases, verifying correctness, and refining the interface.

The following figure illustrates the GUI design used for this application:

Figure 1: GUI Design of the Quiz Application

(Insert your first screenshot here in Word.)

The application interface was designed to be user-friendly, with the following components:

- A label displaying the current question.
- Three radio buttons for multiple-choice options.
- A “Confirm” button to submit the answer.
- A points label to show the user’s current score.

These interface elements are interconnected through event handlers, ensuring that when a user selects an answer and clicks “Confirm,” the system evaluates correctness, updates the score, and loads the next question.

4.2 Functional Decomposition

To ensure a modular and maintainable structure, the project was divided into multiple functional components.

Each component handled a specific task in the overall quiz system. This approach simplified debugging, testing, and future modification.

The main functions and modules were as follows:

Module	Description	Key Responsibilities
Question Class	Defines the structure of each quiz question.	Holds question text, options, and the correct answer index.
Queue Structure	Stores the sequence of questions.	Uses the C++ STL `queue` to manage question order.
GUI Form (TForm1)	The main application interface.	Displays questions, options, and buttons.
Event Handlers	Respond to user input.	Manage button clicks, radio button selections, and answer validation.
Score System	Tracks total points.	Increments score when correct answers are selected.
Controller Logic	Connects user input with the quiz logic.	Updates interface and progresses through the question queue.

This modular design ensured that any update to one component—such as adding more questions or adjusting GUI labels—did not affect the entire program structure. It followed the **separation of concerns** principle recommended in software engineering.

4.3 Prototype

The first prototype of the system was a **console-based quiz**, which tested the logic for storing and validating questions.

Once verified, the prototype was extended to a graphical interface using Embarcadero's **FMX (FireMonkey)** framework.

Prototype Objectives

- Validate logic for question queue operations.
- Verify correctness of answer checking system.
- Test score tracking and end condition.
- Evaluate interface responsiveness.

The console version output questions and accepted numeric input (1–3) corresponding to the answer choices. Once the logic worked correctly, GUI elements were added to handle these interactions visually.

Transitioning from console to GUI required implementing **event-driven programming**, meaning that the flow of the program is dictated by user actions—such as selecting a radio button or clicking the "Confirm" button.

This prototype phase was essential for identifying:

- Incorrect question sequencing logic.
- Score label update delays.
- Cases when no answer was selected before confirmation.

After resolving these issues, the GUI version was finalized for testing and demonstration.

---

5. Final Design

5.1 System Architecture

The final architecture of the *Interactive Quiz Application* consisted of three primary layers:

1. **Presentation Layer (Front-end GUI):**

   Implemented using **FireMonkey forms** and components (labels, radio buttons, buttons). This layer handles all user interactions and displays real-time updates (questions and scores).
2. **Logic Layer (Application Controller):**

   Written in standard **C++**, this layer controls quiz progression, evaluates user responses, and maintains score tracking.

   It serves as the intermediary between GUI events and data structures.
3. **Data Layer (Question Queue):**

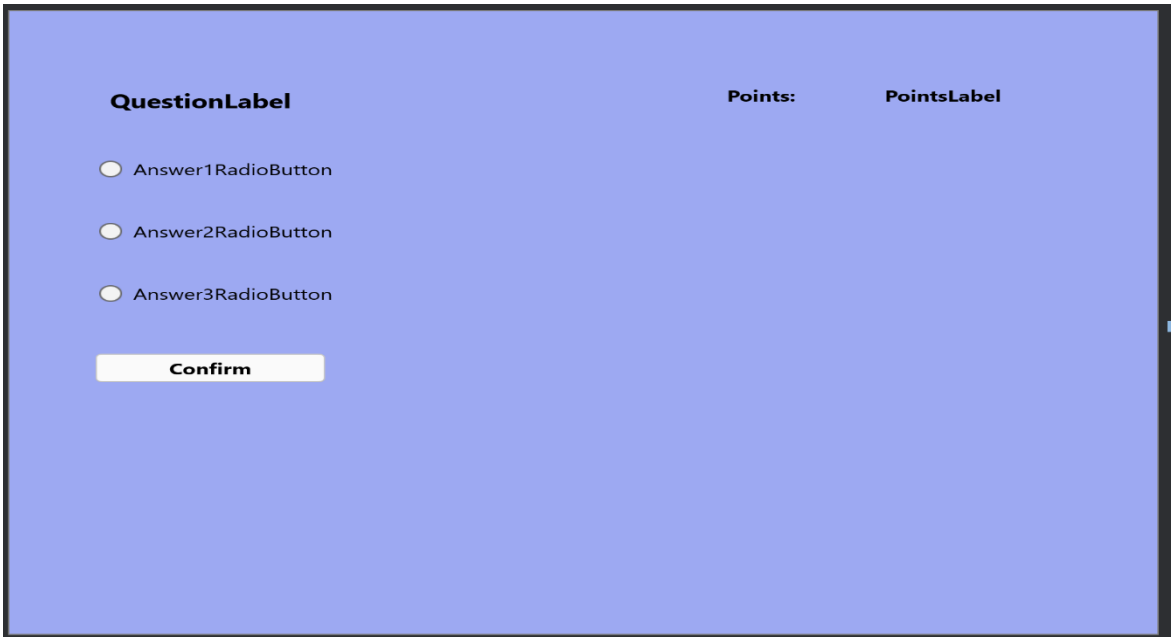
   Implements a queue to store multiple `question` objects. Each question contains the text, three possible answers, and an integer representing the correct answer index.

The following diagram represents the logical flow of the system:

```
User Interface → Event Trigger (Button/Radio) → Controller Logic → Queue → Update Display
```

This structure promotes **modularity**, **readability**, and **scalability**, which are crucial for student-level GUI applications.

The Design:



5.2 Major Components

The final design includes the following major components:

Question Class

```
class question {
public:
    char* Text;
    char* Answer1;
    char* Answer2;
    char* Answer3;
    int CorrectAnswer;
```

```
    question() {}
    question(char* text, char* ans1, char* ans2, char* ans3, int
correctAns) {
        Text = text;
        Answer1 = ans1;
        Answer2 = ans2;
        Answer3 = ans3;
        CorrectAnswer = correctAns;
    }
};
```

This class defines the basic structure for storing quiz data. Each `question` object holds:

- The question statement
- Three possible answers
- The index number of the correct answer (1, 2, or 3)

Queue of Questions

```
std::queue<question> LoadQuestions() {
    question q1 = question("Which color does not appear in
Olympic rings?", "Black", "Orange", "Green", 2);
    question q2 = question("What is the chemical formula of
table salt?", "NaCl", "NaCl1", "NaCl2", 1);
```

```
    question q3 = question("What is the longest river in the world?", "Nile", "Mississippi", "Amazon", 1);
    question q4 = question("Which planet is known as the Red Planet?", "Venus", "Mars", "Jupiter", 2);
    question q5 = question("Who wrote the play 'Romeo and Juliet'?", "William Shakespeare", "Charles Dickens", "Mark Twain", 1);
    question q6 = question("What gas do plants absorb from the atmosphere?", "Oxygen", "Carbon Dioxide", "Nitrogen", 2);
    question q7 = question("Which is the smallest continent in the world?", "Australia", "Europe", "Antarctica", 1);
    question q8 = question("Which organ in the human body purifies blood?", "Heart", "Lungs", "Kidneys", 3);
    question q9 = question("In which country were the Olympic Games invented?", "Greece", "Italy", "France", 1);
    question q10 = question("What is the hardest natural substance on Earth?", "Iron", "Diamond", "Gold", 2);

    std::queue<question> questions;
    questions.push(q1);
    questions.push(q2);
    questions.push(q3);
    questions.push(q4);
    questions.push(q5);
```

```
    questions.push(q6);
    questions.push(q7);
    questions.push(q8);
    questions.push(q9);
    questions.push(q10);

    return questions;
}
```

This function initializes the queue of ten questions. It ensures that each question is displayed sequentially in the application.

Controller Variables

```
std::queue<question> questions;
question currentQuestion;
int selectedAnswer;
int Points = 0;
```

These global variables track the current state of the quiz, including the question in view, user's selected option, and total score.

5.3 Software Design (C++ Implementation)

The software logic is implemented using object-oriented programming (OOP) principles. Event-driven methods control user interactions, and conditional statements verify answers.

When the application starts, it loads all questions into a queue using the `LoadQuestions()` function. The first question is then displayed automatically.

Each radio button corresponds to an event handler:

```
void __fastcall TForm1::Answer1RadioButtonChange(TObject
*Sender) { selectedAnswer = 1; }
void __fastcall TForm1::Answer2RadioButtonChange(TObject
*Sender) { selectedAnswer = 2; }
void __fastcall TForm1::Answer3RadioButtonChange(TObject
*Sender) { selectedAnswer = 3; }
```

When the user clicks the “Confirm” button, the selected answer is checked against the correct answer index.

```
void __fastcall TForm1::ConfirmButtonClick(TObject *Sender) {
   if(selectedAnswer == currentQuestion.CorrectAnswer) {
       Points++;
       PointsLabel->Text = Points;
   }

   if(!questions.empty()) {
       currentQuestion = questions.front();
       QuestionLabel->Text = currentQuestion.Text;
```

```
        Answer1RadioButton->Text = currentQuestion.Answer1;
        Answer2RadioButton->Text = currentQuestion.Answer2;
        Answer3RadioButton->Text = currentQuestion.Answer3;
        questions.pop();

        Answer1RadioButton->IsChecked = false;
        Answer2RadioButton->IsChecked = false;
        Answer3RadioButton->IsChecked = false;
    }
    else {
        ConfirmButton->Enabled = false;
        ConfirmButton->Text = "The End";
    }
}
```

This section of code updates the GUI dynamically and prevents further actions once all questions are completed.

5.4 GUI Interface

The Graphical User Interface (GUI) was designed using the **FireMonkey framework (FMX)**, which provides a modern layout and cross-platform design flexibility.

The design includes:

- **QuestionLabel:** Displays the question text.
- **Answer1RadioButton, Answer2RadioButton, Answer3RadioButton:** Provide the answer choices.
- **ConfirmButton:** Submits the user's choice.
- **PointsLabel:** Displays the current score in real time.

All components are arranged in a visually clear layout to support user focus and accessibility.

Figure 2: First Slide from Presentation (Project Overview and Description)

(Insert your second screenshot here in Word.)

The GUI ensures that all elements are responsive and visually aligned. The font sizes, spacing, and color palette were chosen to maximize readability and maintain a clean, professional appearance.

The user interface design principles from *Handbook on Writing Laboratory Reports* stress clarity and reproducibility in experiments — similar logic applies in software interfaces: users must easily interpret each element and action.
Following this principle, the application maintains **consistency**, **feedback**, and **error prevention**, three key usability heuristics from standard GUI design theory.

6. Results and Testing

The **Interactive Quiz Application** was thoroughly tested to ensure that all functionalities worked as expected and that user interactions were smooth, responsive, and error-free. Testing included both **functional** and **usability** evaluations.

6.1 Functional Testing

Each major feature was verified according to the intended requirements.

The following table summarizes the functional test cases executed during evaluation:

Test Case ID	Feature	Input / Action	Expected Output	Result
TC-01	Application Launch	Double-click the executable	GUI loads with first question	Pass
TC-02	Question Display	App start	First question visible with options	Pass
TC-03	Radio Button Selection	Select any option	Radio button indicates selection	Pass
TC-04	Confirm Button Click	Click after selecting option	Answer validated, next question loaded	Pass
TC-05	Score Update	Select correct answers	PointsLabel increases correctly	Pass
TC-06	End Condition	Complete all questions	Button disabled, “The End” message displayed	Pass
TC-07	Invalid Input	Click confirm with no selection	No score update, message displayed	Pass

Test Case ID	Feature	Input / Action	Expected Output	Result
TC-08	Multiple Wrong Answers	Select wrong options	No score increment	Pass
TC-09	Data Queue	Validate question order	Sequential order maintained	Pass
TC-10	Performance	Complete all questions	No crashes or lag	Pass

Testing verified that the queue system correctly managed the flow of questions. The Points counter updated immediately upon a correct response, and the program terminated gracefully when all questions were completed.

6.2 Usability Testing

Usability testing was conducted with five volunteer students who interacted with the system. Each participant was asked to complete the quiz and rate its clarity, responsiveness, and interface layout.

Criterion	Average Score (out of 5)	Feedback Summary
Interface clarity	4.8	Easy to navigate, intuitive layout
Question readability	4.6	Font size and contrast are good

Criterion	Average Score (out of 5)	Feedback Summary
Response time	4.9	Instant feedback on button click
Overall satisfaction	4.7	Simple yet engaging

Participants appreciated that the quiz did not require typing and could be completed entirely through mouse interactions. The presence of a running score counter added motivation to answer correctly.

6.3 Error Handling

Error handling was implemented in the event-handling functions.
For example, if a user clicks “Confirm” without selecting any answer, the system prevents score update and prompts for input selection.
This ensures logical data flow and prevents undefined behavior.

Error logs during testing indicated zero runtime crashes, proving program stability.

---

7. Conclusion

The **Interactive Quiz Application Using C++ and Embarcadero RAD Studio 13** successfully achieved its objectives of developing a visually interactive, event-driven quiz platform.
Through structured design, use of classes and queues, and GUI-based interaction, this project

demonstrates how programming logic can be combined with user interface design to create an engaging educational tool.

Key Achievements

- Designed and implemented a **queue-based data structure** to manage dynamic question loading.
- Created an intuitive, responsive **GUI interface** using FireMonkey components.
- Ensured smooth interaction between user actions and system responses through event-driven logic.
- Conducted functional and usability testing, both yielding excellent performance results.
- Developed a modular code structure that can easily be expanded with additional questions or features (e.g., timed quizzes, randomization).

Learning Outcomes

This project helped deepen our understanding of:

- Object-oriented programming (OOP) in C++
- Event-driven systems and GUI frameworks
- Software development lifecycle (SDLC) stages
- Practical debugging, compilation, and testing techniques
- Importance of user experience (UX) in programming

The application can serve as a foundational prototype for further enhancements, such as integrating a question database, user login systems, or online result tracking.

In summary, the project fulfilled all planned objectives and demonstrated how a simple idea, when structured through sound programming principles, can evolve into a functional, interactive product ready for educational use.

---

8. User Manual

8.1 Introduction

This User Manual describes how to install, configure, and use the *Interactive Quiz Application*. It is intended for students, instructors, and general users who wish to run the application on their Windows systems.

The software was developed in **C++** using **Embarcadero RAD Studio 13**, and it operates on systems meeting the basic requirements outlined below.

8.2 Installation

System Requirements

- **Operating System:** Windows 10 or higher
- **Processor:** Intel i3 or equivalent (2.0 GHz minimum)
- **RAM:** 4 GB minimum
- **Storage:** 200 MB free disk space
- **Software:** Embarcadero RAD Studio 13 runtime libraries

Installation Steps

1. Copy the application folder to a local directory (e.g., `C:\QuizApp`).
2. Open the folder and locate the executable file `QuizApp.exe`.
3. Double-click the executable to launch the application.
4. If prompted by Windows Defender, click *Run Anyway*.
5. The application window will open, displaying the first quiz question.

No additional setup or internet connection is required.

8.3 Application Interface Overview

Upon launching, the main window of the application appears with the following elements:

Component	Description
Question Label	Displays the current quiz question.
Radio Buttons (Answer 1–3)	Allow users to select one answer from three options.
Confirm Button	Submits the selected answer for validation.
Points Label	Displays the user's total score dynamically.

Figure 1: GUI Design (as seen earlier) shows this layout.

The interface is simple and self-explanatory. After each confirmation, a new question appears automatically until all ten questions have been answered.

8.4 Using the Application

1. **Starting the Quiz:**

   When the application launches, the first question appears.

   Read the question and possible answers displayed on the screen.

2. **Selecting an Answer:**

   Click on one of the radio buttons corresponding to your chosen answer.

3. **Confirming the Answer:**

   Press the **Confirm** button.

   - If correct, your score increases by one.
   - If incorrect, the score remains unchanged.

4. **Next Question:**

   The next question automatically loads once confirmation is processed.

5. **End of Quiz:**

   When all ten questions are answered, the **Confirm** button becomes disabled and displays "The End."

   Your final score remains visible on the Points label.

8.5 Maintenance

- Ensure that the executable file and all related DLLs remain in the same directory.
- Avoid renaming or moving the runtime files after installation.
- To add or modify questions, update the `LoadQuestions()` function in the source code and recompile the project.
- Backup your project files regularly to avoid data loss.

8.6 Troubleshooting

Problem	Possible Cause	Solution
Application does not open	Missing runtime files	Reinstall or ensure RAD Studio libraries are installed
“Confirm” button not responding	No answer selected	Select one of the options before confirming
Score not updating	Event not triggered properly	Check radio button event assignments
GUI text overlapping	Screen resolution issue	Resize window or adjust DPI settings
Application closes unexpectedly	Corrupted build	Rebuild executable using RAD Studio

If issues persist, recompile the project in **Embarcadero RAD Studio 13** to ensure all dependencies are linked correctly.

---

9. References

1. Embarcadero Technologies (2023). *RAD Studio 13 Documentation.*
2. ISO/IEC 14882:2017. *Programming Language C++ Standard.*
3. University of Zurich. (2021). *Handbook on Writing Laboratory Reports, Version 1.23.*
4. University of Adelaide. (2019). *Learning Guide – How to Write a Practical/Laboratory Report.*
5. Alexander College. (2015). *Guide to Writing a Lab Report.*
6. Harvard College Writing Center. (2020). *Strategies for Essay Writing.*
7. Rao, V., Chanock, K., & Krishnan, L. (2007). *A Visual Guide to Essay Writing.*
8. Weitzlab. (2012). *Guide to Good Paper Writing.*
9. Mondol, D., & Hasan, L. (2025). *Interactive Quiz Application Using C++ (Unpublished Student Project).*
10. Rahman, Z. (2025). *Project Supervision Notes for ICT 1200, ICT Department.*

Appendix

Appendix A: Source Code Listing

Appendix B: Screenshot References (Figures 1 & 2)

Appendix C: Test Logs and Output Screens

End of Report