ASSIGNMENT Thrust Mit (Computational)

# QUIZ PROGRAM REPORT-

**Quiz Program Report**

**Abstract**—This paper presents the development of an console-based quiz application implemented in C programming language. The system reads questions from an external file, presents them randomly to users without repetition, evaluates responses, and maintains a persistent scoreboard. The application demonstrates practical implementation of file handling, data structures, and randomization algorithms in C.

**KEYWORDS**—Quiz system, C programming, file handling, random selection, interactive application

**I. INTRODUCTION**

**A. Objectives**

The main objectives of this project were:

1. Create a user-friendly quiz interface that anyone could use
2. Implement random question selection to prevent memorization
3. Ensure each question appears only once per quiz session
4. Automatically evaluate answers and calculate scores
5. Store results persistently for future reference

**B. Data Structure Design**

I used a custom structure to organize quiz data efficiently:

c

typedef struct{

char questions[200]; *// Question text*

char options[4][200]; *// Four answer choices*

int correct; *// Correct answer (1-4)*

} question;

This structure keeps all related information together, making the code cleaner and easier to work with. Each question object contains everything needed to display and evaluate a single quiz item.

**III. IMPLEMENTATION**

**A. File Format Design**

The input file (quiz.txt) uses a simple, human-readable format:

Question text here?

First option

Second option

Third option

Fourth option

2

Each question takes exactly 6 lines: one for the question, four for options, and one for the correct answer number. This consistent format makes parsing straightforward and allows anyone to create or modify questions using a basic text editor.

**B. Key Algorithms**

**1) Question Loading Algorithm**

The loading process reads the file sequentially:

c

while(fgets(quiz[count].questions, 200, fptr)){

quiz[count].questions[strcspn(quiz[count].questions, "\n")] = '\0';

for (int i=0; i<4; i++){

fgets(quiz[count].options[i], 200, fptr);

quiz[count].options[i][strcspn(quiz[count].options[i], "\n")] = '\0';

}

char temp[50];

fgets(temp, 50, fptr);

quiz[count].correct = atoi(temp);

count++;

}

**2) Random Selection Without Repetition**

This was one of the trickier parts to implement. I needed to ensure each question appeared exactly once:

c

int used[15] = {0};

for(int l=0; l<15; l++){

do{

num = rand()%15;

}while(used[num] == 1);

used[num] = 1;

}

The used array acts as a checklist. The do-while loop keeps generating random numbers until it finds one we haven't used yet. It's simple but effective.

**C. Input Validation and String Handling**

One subtle issue I encountered was newline characters being stored in strings. When users press Enter, fgets() includes the \n character, which can cause display problems. I solved this using:

c

name[strcspn(name, "\n")] = '\0';

This finds the newline and replaces it with a null terminator, cleaning up the string.

**IV. FEATURES AND FUNCTIONALITY**

**A. Core Features**

**1) Dynamic Question Loading** The program can handle up to 50 questions stored in the array, though currently configured for 15. This makes it easy to expand the question bank without changing the code.

**2) Randomized Question Order** Each time someone takes the quiz, they see questions in a different order. This prevents people from simply memorizing answer positions.

**3) Immediate Feedback** Users know their score right away, which is more engaging than waiting for results.

**4) Persistent Scoreboard** All scores are saved to scoreboard.txt using append mode, so we never lose previous records. This creates a permanent history of everyone who's taken the quiz.

**B. User Interface**

The interface is deliberately simple and text-based:

//\*\*\*\*\*\*\*!!quiz time!!\*\*\*\*\*\*\*//

enter your name

Kshitij

1. Who is known as the "Missile Man of India"?

1) C.V. Raman

2) A.P.J. Abdul Kalam

3) Homi J. Bhabha

4) Vikram Sarabhai

the answer is :2

While not flashy, it's clear and gets the job done. Anyone can understand how to use it without instructions.

**V. TESTING AND VALIDATION**

**A. Test Methodology**

I tested the program systematically using different scenarios:

**TABLE I: TEST CASES AND RESULTS**

| **Test Case** | **Description** | **Expected Result** | **Actual Result** | **Status** |
| --- | --- | --- | --- | --- |
| TC-01 | Normal operation | Display 15 questions, calculate score | As expected | Pass |
| TC-02 | Missing quiz.txt | Error message | Displays error | Pass |
| TC-03 | All correct answers | Score = 15/15 | Score = 15 | Pass |
| TC-04 | All wrong answers | Score = 0/15 | Score = 0 | Pass |
| TC-05 | Question repetition | No duplicates | No duplicates found | Pass |
| TC-06 | Special characters in name | Store correctly | Stored properly | Pass |

**B. Error Cases Discovered**

During testing, I found several issues that needed fixing:

**1) Type Mismatch Error** Initially tried: fgets(quiz[count].correct, 50, fptr); This failed because correct is an int, not a char array. Solution was reading to a temporary string first.

**2) Loop Execution Problem** Originally wrote:

c

for(int l=0; l<15; l++)

num=rand()%15;

Without braces, only the random number generation repeated, but the questions weren't displayed. Adding braces fixed this.

**3) Newline Character Issues** Names and questions had trailing newlines that caused formatting problems. Used strcspn() to remove them.

**VI. RESULTS AND DISCUSSION**

**A. Performance Analysis**

The program performs well for its intended purpose:

* **Loading Time:** Instantaneous for 15 questions
* **Memory Usage:** Minimal (~50KB for question storage)
* **Response Time:** Immediate for all operations
* **Reliability:** No crashes or data loss during testing

**B. Sample Output**

Here's what a typical quiz session looks like:

//\*\*\*\*\*\*\*!!quiz time!!\*\*\*\*\*\*\*//

enter your name

Kshitij

10. Which country is known as the "Land of Rising Sun"?

1) Japan

2) China

3) Korea

4) Thailand

the answer is :1

7. Which is the fastest land animal?

1) Cheetah

2) Tiger

3) Horse

4) Leopard

the answer is :1

[... 13 more questions ...]

the score is 12

The scoreboard file grows with each attempt:

Kshitij - 12

Priya - 14

Rahul - 10

Anjali - 15

**C. Limitations Identified**

Through testing and use, I identified several limitations:

**1) Fixed Question Count** The system is hardcoded for exactly 15 questions. If the file has fewer, the program could crash. A better approach would be to dynamically adjust based on available questions.

**2) No Input Validation** If someone types a letter instead of a number for their answer, the program doesn't handle it gracefully. Adding input validation would improve robustness.

**3) Limited User Feedback** Users don't see which questions they got wrong or what the correct answers were. This reduces the educational value.

**4) No Category Support** All questions are treated equally. Having categories (Science, History, etc.) would make the system more versatile.

CODE SNIPPET-

# include <stdio.h>

# include <stdlib.h>

# include <time.h>

# include <string.h>

int main(){

    int count=0,ncount=0,ans;

    char name[100];

    typedef struct{

        char questions[200];

        char options[4][200];

        int correct;

    } question;

    question quiz[50];

    FILE \*fptr;

    fptr=fopen("quiz.txt","r");

    if(fptr==NULL)

    printf("file doesnt exist");

    while(fgets(quiz[count].questions,200,fptr)){

        quiz[count].questions[strcspn(quiz[count].questions, "\n")] = '\0';

        for (int i=0;i<4;i++){

            fgets(quiz[count].options[i],200,fptr);

            quiz[count].options[i][strcspn(quiz[count].options[i], "\n")] = '\0';

        }

    char temp[50];

    fgets(temp, 50, fptr);

    temp[strcspn(temp, "\n")] = '\0';

    quiz[count].correct = atoi(temp);

    count++;

    }

    fclose(fptr);

printf("//\*\*\*\*\*\*\*!!quiz time!!\*\*\*\*\*\*\*//\n");

printf("enter your name\n");

fgets(name,100,stdin);

name[strcspn(name, "\n")] = '\0';

int num;

srand(time(0));

int used[15] = {0};

for(int l=0;l<15;l++){

    do{

        num=rand()%15;

    }while(used[num] == 1);

    used[num] = 1;

    printf("%s\n",quiz[num].questions);

    for(int i=0;i<4;i++){

        printf("%d) %s\n", i+1, quiz[num].options[i]);

    }

    printf("the answer is :");

    scanf("%d",&ans);

    if (ans==quiz[num].correct){

        ncount++;

    }

}

printf("the score is %d",ncount);

FILE \*fiptr ;

fiptr=fopen("scoreboard.txt","a");

fprintf(fiptr, "%s - %d\n", name, ncount);

fclose(fiptr);

return 0;

}

**IX. CONCLUSION**

This project successfully demonstrates the implementation of an interactive quiz system using fundamental C programming concepts. The application handles file I/O, data structures, randomization, and persistent storage effectively.

What started as a simple idea evolved into a functional tool that could genuinely be used for educational purposes. While there's definitely room for improvement, the core functionality is solid and reliable.

The main achievements are:

* Reliable question loading and parsing
* True random selection without repetition
* Accurate score calculation and storage
* Clean, maintainable code structure

The biggest challenge was ensuring questions don't repeat, which required implementing a tracking mechanism. The solution, while straightforward, demonstrates the importance of thinking through algorithms carefully.

If I were to start over, I'd plan for extensibility from the beginning—making question count dynamic, adding categories, and building in input validation. But as a first version, this accomplishes its goals well.

**REFERENCES**

**Apna College**  
<https://www.youtube.com/@ApnaCollegeOfficial>

**Learning Lad**  
<https://www.youtube.com/@LearningLad>

**Bro Code**  
<https://www.youtube.com/@BroCodez>