Republic of the Philippines
RIZAL TECHNOLOGICAL UNIVERSITY
Cities of Mandaluyong and Pasig
COLLEGE OF ENGINEERING, ARCHITECHTURE AND TECHNOLOGY

COMPUTER PROGRAMMING 1

(ITC111L)

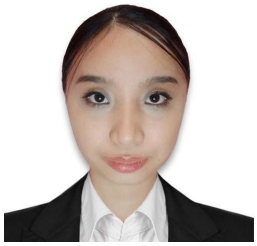
QUIZ WIZZARD

GRADE

Submitted by:

Morales, Jade BSIT CEIT-37-202A

Almazan, Janna Ricci BSIT CEIT-37-202A



Lugod, Brendan Jane BSIT CEIT-37-202A

Bondad, John Christopher BSIT CEIT-37-202A

Advincula, Chino Clarence BSIT CEIT-37-202A

Depra, Dewell Helsen BSIT CEIT-37-202A

Misagal, Edgar Allan BSIT CEIT-37-202A

Francis, Mark Ghian BSIT CEIT-37-202A

Submitted to:

Prof. May Barcelona Figueroa

Professor May 26, 2023

I. INTRODUCTION

Welcome to our C++-based quiz game software! The purpose of this project is to give people a fun and interactive way to learn new material while also testing their knowledge.We made the decision to start this initiative for a variety of reasons. First of all, we acknowledged the appeal of quiz games and the fun that they provide for players of all ages. We intended to design an application that might provide a comparable experience, giving users a place to test their limits and compete with one another. Quizzes, in our view, are a great approach to improve knowledge acquisition and memory. Our application acts as a useful tool for knowledge development and self-improvement by including instructional questions from a range of areas. We want to create a quiz that is both entertaining and educational.During the course of the development process, we encountered some challenges to overcome. Making a user-friendly, intuitive interface that can accommodate users of all ability levels was one of the key challenges. We wanted to make sure that users of all skill levels could use the application with ease and enjoy taking quizzes. We thoroughly studied C++ programming procedures and made use of our knowledge of data structures and algorithms to overcome these difficulties. To ensure the dependability and stability of our application, we have created a strict testing and debugging procedure.Overall, we want users to have a smooth and delightful experience while being entertained, educated, and challenged by our quiz game program.

II. DESCRIPTIONOFTHEPROJECT

The researcherhas developed a Quiz Game project that offers various game modes upon compiling and running the source code. These modes include classic, time trial, pass and play, and survival. In the classic mode, there are randomly multiple-choice or identification questions consisting of 10 unique questions. Players must answer all the questions correctly to pass the game. The time trial mode challenges players to answer each question within a 60-second time limit. If a player fails to respond within the given time frame, the game automatically moves on to the next question. Pass and play mode is designed for multiplayer gameplay with two participants. Prior to starting the game, players enter their names. The first player to achieve a score of 10 emerges as the winner. Once the game ends, the results display which player has secured the victory. In survival mode, players begin with 4 lives and are tasked with answering 10 questions in each level. If a player loses, a message appears indicating "you lost the game," along with the player's remaining lives and score. Conversely, if a player emerges victorious, a message is displayed stating "you won the game." During the "Add Question" process, users can add questions to all available modes. They have the option to choose whether the questions will
be in multiple-choice format or not. Furthermore, users are prompted to input the desired number of questions. Additionally, the program features a menu that offers access to various options, including credits, information about the game, its benefits, documentation, and the game's version.

.

III. OBJECTIVES

The main objective of this system is to provide an alternative learning platform thatc

onsists of different modes and features to help users in their studies, especially in their

programming examinations. and our system objective includes:

1) To make aprogram use C++ that allows users to assess themselves using this system.

2) To use programming techniques like vector, filehandling, conio.h, cstdlib, string.h, stdio.h, windows. h and essential.h

IV. SIGNIFICANCE OF THE STUDY

The significance of our program resides in the fact that quiz games encourage students' self-awareness and self-evaluation. By completing quizzes, students may easily get feedback on their responses, identifying their strengths and weaknesses and highlighting their accomplishments.The difficulty levels, time constraints, multiple choice questions, different kinds of questions, and feedback are the most crucial information in this work. Students can increase their speed and accuracy by using varied difficulty levels, time constraints, multiple choice questions, different sorts of questions, and feedback.

V. SCOPE AND DELIMITATIONS

This C++ program is designed to make an interactive Quizz system that provides the user choices with different mode options. This program is focused on giving the users their remark through numbering system that they could access in the attempt history, this program covers features such as:

- Usersign-up
- Userattempthistory
- Number System Delimitations

In order to maintain focus and ensure the feasibility of the system, certain delimitations have been established. These delimitations outline the boundaries and limitations within which the study will be conducted. The delimitations of this research include:

- InPassandPlay,two(2)userscanaccessthismodeusingonedeviceonly.
- In Custom Write, users cannot exit or cancel the process during

creation of questions and options.

VI. SCREENOUTPUT

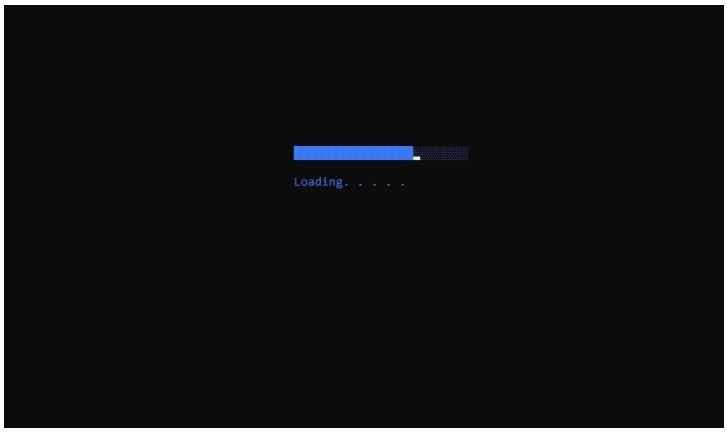


Figure 1. Loading Screen

Figure 1 shows the loading of the screen before the Quiz Wizzard starts

Figure 2. Menu Options

Figure shows the menu option:
Start Quiz - Shows the modes of the quizzes
Instruction - shows the instructions on how to play
Game History- shows the previous plays the user have done in the history
Custom Question- lets the user make their own questionnaire
Exit - exits the program /system

User can type his/ her beside the word "choice:"

Figure 3. Mode option

Figure 3 shows the mode options whereas:

Classic - will let the user choose the quiz type whether identification or multiple choice
Time Trial- in a multiple choice setting user will have to answer the questions inside the given time only
Pass and play- lets the user play with a friend (multiplayer)
Survival- user has the answer the multiple choice correctly or its lives will be redacted ones wrong answer
Menu - leads the user back to the main menu

Figure 4. Game Instructions

Figure 4 Shows the instructions of all the game modes

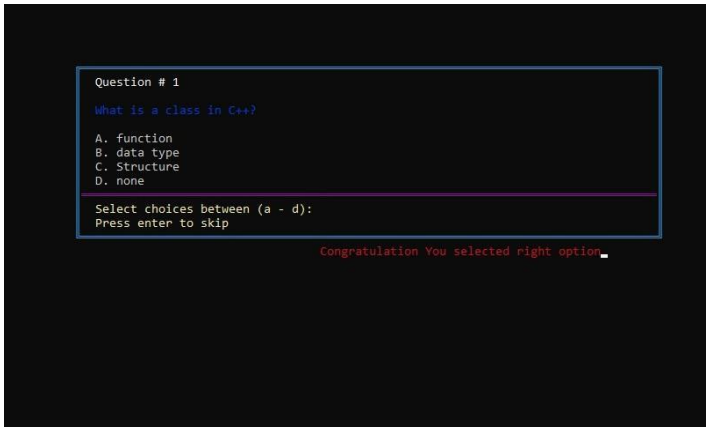


Figure 5. Classic Mode

Figure 5 shows questions in classic mode. For each correct answer, your score will increase by 1 point

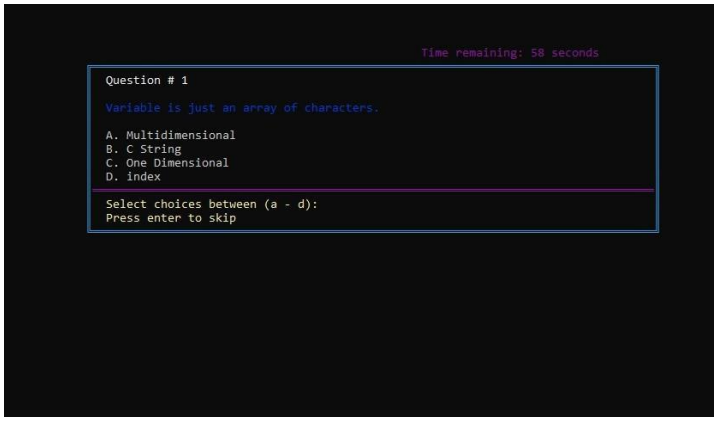


Figure 6. Time Trial Mode

Figure 6 shows the questions for time trial mode. If time runs out and all questions are not answered, the user loses.

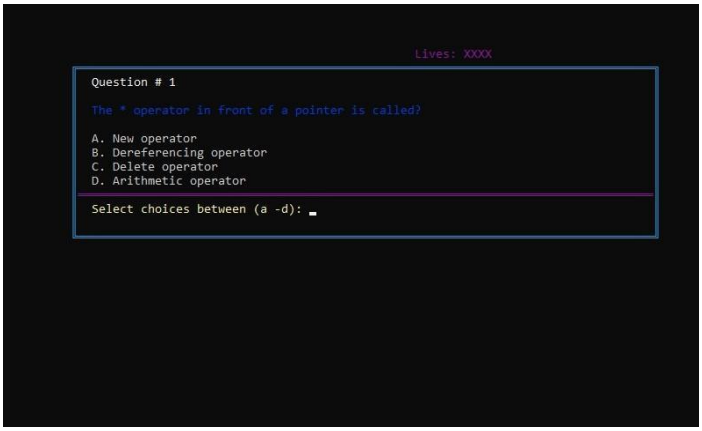


Figure 7. Survival Mode

Figure 7 shows the questions for survival mode. The user will be given 4 lives. Each wrong answer costs 1 life

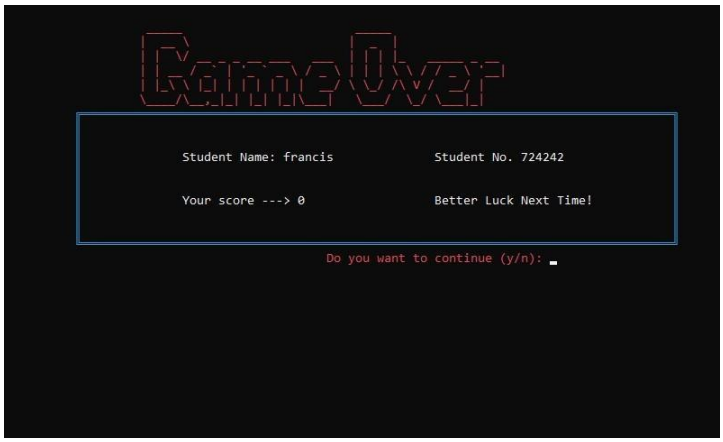


Figure 8. Game Over Screen

Figure 8 shows the game over screen. This screen will show when the user loses a game.

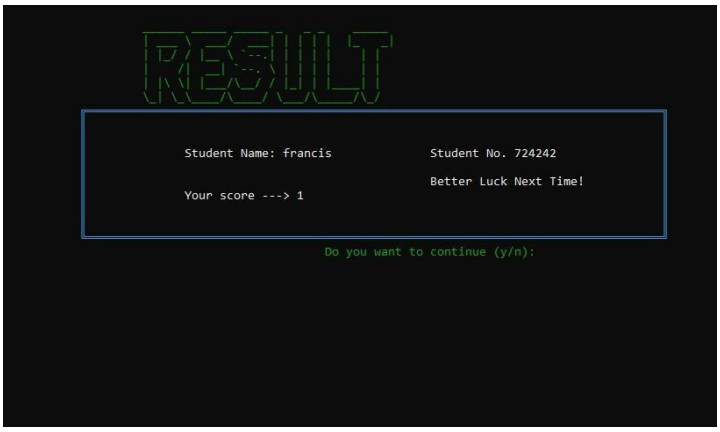


Figure 9. Result Screen

Figure 9 shows the user's score.

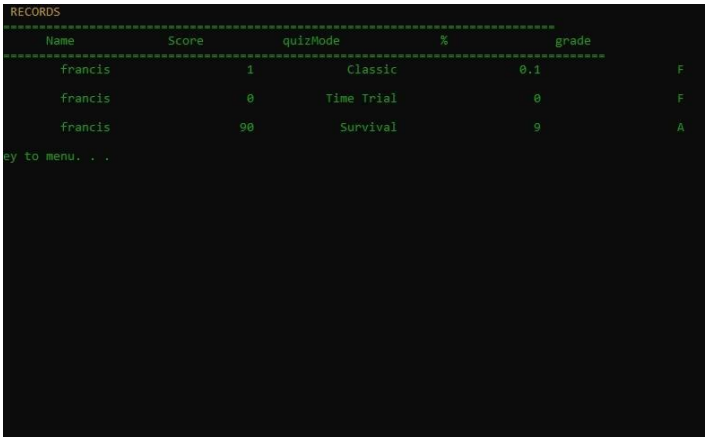


Figure 10. Game History

Figure 10 shows the scores of the users for each mode with their name.

Figure 11. Custom Mode

In Figure 11, you can add your own questions and choices.

VII. SOURCECODE

MAIN MENU (SOURCE CODE)

```
#include <iostream>
#include "classicMode.h"
#include "passAndPlay.h"
#include "timeTrialMode.h"
#include "survivalMode.h"
#include "essential.h"
#include "userGameHistory.h"
#include "addQuestion.h"

//This class is responsible for mainmenu
void selectMenu();
void titleScreen();
HANDLE h = GetStdHandle(STD_OUTPUT_HANDLE);

class MainMenu
{
private:
   int userChoice{};
   int choice{};

public:

   void mainMenu()
   {
      essential::gotoXY(50, 8);
      SetConsoleTextAttribute(h, 15);
      std::cout << "=-=-=-=- Select -=-=-=-=";
      essential::gotoXY(55, 10);
      SetConsoleTextAttribute(h, 15);
      std::cout << "1--> Start quiz";
      essential::gotoXY(55, 12);
      SetConsoleTextAttribute(h, 15);
      std::cout << "2--> Instruction";
      essential::gotoXY(55, 14);
      SetConsoleTextAttribute(h, 15);
      std::cout << "3--> Game History";
      essential::gotoXY(55, 16);
      SetConsoleTextAttribute(h, 15);
      std::cout << "4--> Custom Question";
      essential::gotoXY(55, 18);
      SetConsoleTextAttribute(h, 15);
      std::cout << "5--> Exit";
      essential::gotoXY(55, 20);
      SetConsoleTextAttribute(h, 15);
      std::cout << "Choice: ";
   }

   void selectMode()
   {
      classicMode::ClassicMode classic;
      passAndPlay::PassAndPlay pAndP;
      timeTrialModes::TimeTrialMode timeMode;
```

```
    survivalModes::SurvivalMode surMode;

    system("cls");
    titleScreen();
    dogDesign();

    essential::gotoXY(50, 8);
    SetConsoleTextAttribute(h, 15);
    std::cout << "=-=-=-=- Select Mode -=-=-=-=";
    essential::gotoXY(56, 10);
    SetConsoleTextAttribute(h, 15);
    std::cout << "1--> Classic";
    essential::gotoXY(56, 12);
    SetConsoleTextAttribute(h, 15);
    std::cout << "2--> Time Trial";
    essential::gotoXY(56, 14);
    SetConsoleTextAttribute(h, 15);
    std::cout << "3--> Pass and play";
    essential::gotoXY(56, 16);
    SetConsoleTextAttribute(h, 15);
    std::cout << "4--> Survival";
    essential::gotoXY(56, 18);
    SetConsoleTextAttribute(h, 15);
    std::cout << "5--> Menu ";
    essential::gotoXY(56, 20);
    SetConsoleTextAttribute(h, 15);
    std::cout << "choice: ";
    std::cin >> choice;

    switch (choice)
    {
      case 1:
        classic.startGame(&selectMenu);
        break;

      case 2:
        timeMode.startQuiz(&selectMenu);
        break;

      case 3:
        pAndP.startGame(&selectMenu);
        break;

      case 4:
        surMode.startQuiz(&selectMenu);
        break;

      default:
        //if the user select the wrong input it will start in the begining of selecting mode
        selectMode();
        break;
    }
}

void gameInstruction()
{
    system("cls");
    essential::gotoXY(56,4);
```

```
 SetConsoleTextAttribute(h, 11);
  std::cout << "Game Instructions";
  essential::gotoXY(50,5);

  SetConsoleTextAttribute(h, 10);
essential::gotoXY(38,6);
          std::cout<<"  _____"<< '\n';
          essential::gotoXY(38,7);
          std::cout <<" / \\                              \\."<< '\n';
          essential::gotoXY(38,8);
          std::cout <<"|   |                             |."<< '\n';
          essential::gotoXY(38,9);
          std::cout <<" \\_ | Select==>                    |."<< '\n';
          essential::gotoXY(38,10);
          std::cout <<"    | Press only Valid Opton--> (a,b,c,d)    |."<< '\n';
          essential::gotoXY(38,11);
          std::cout <<"    | if u Press Other key consider wrong    |."<< '\n';
          essential::gotoXY(38,12);
          std::cout <<"    | answer.                      |."<< '\n';
          essential::gotoXY(38,13);
          std::cout <<"    |                                |."<< '\n';
          essential::gotoXY(38,14);
          std::cout <<"    | Skip==>                      |."<< '\n';
          essential::gotoXY(38,15);
          std::cout <<"    | Press Enter to Skip the Question       |."<< '\n';
          essential::gotoXY(38,16);
          std::cout <<"    |                             |."<< '\n';
          essential::gotoXY(38,17);
          std::cout <<"    | Points==>                     |."<< '\n';
          essential::gotoXY(38,18);
          std::cout <<"    | 10 Point will be awarded for each      |."<< '\n';
          essential::gotoXY(38,19);
          std::cout <<"    | correct answer.                |."<< '\n';
          essential::gotoXY(38,20);
          std::cout <<"    |                             |."<< '\n';
          essential::gotoXY(38,21);
          std::cout <<"    | Time Trial Mode==>              |."<< '\n';
          essential::gotoXY(38,22);
          std::cout <<"    | Player will be given 60 seconds for    |."<< '\n';
          essential::gotoXY(38,23);
          std::cout <<"    | each question.                 |."<< '\n';
          essential::gotoXY(38,24);
          std::cout <<"    |                             |."<< '\n';
          essential::gotoXY(38,25);
          std::cout <<"    | Survival Mode==>                |."<< '\n';
          essential::gotoXY(38,26);
          std::cout <<"    | Player will be given 4 lives per game  |."<< '\n';
          essential::gotoXY(38,27);
          std::cout <<"    | save                           |."<< '\n';
          essential::gotoXY(38,28);
          std::cout <<"    |  _____|."<< '\n';
          essential::gotoXY(38,29);
          std::cout <<"    |  /                          /."<< '\n';
          essential::gotoXY(38,30);
          std::cout <<"    \\_/_____/."<< '\n';

  getch();
  SetConsoleTextAttribute(h, 15);
```

```
        std::cout << "Press Enter to Home Menu" << '\n';
        selectMenu();
    }

    void dogDesign()
    {
        essential::gotoXY(73, 22);
        SetConsoleTextAttribute(h, 5);
        std::cout <<R"(     |\_/|              )";
        essential::gotoXY(73, 23);
        SetConsoleTextAttribute(h, 5);
        std::cout <<R"(     | @ @   Welcome!!    )";
        essential::gotoXY(73, 24);
        SetConsoleTextAttribute(h, 5);
        std::cout <<R"(     |   <>            _ )";
        essential::gotoXY(73, 25);
        SetConsoleTextAttribute(h, 5);
        std::cout <<R"(     |  _/\------____ ((| |)))";
        essential::gotoXY(73, 26);
        SetConsoleTextAttribute(h, 5);
        std::cout <<R"(     |              `--' |)";
        essential::gotoXY(73, 27);
        SetConsoleTextAttribute(h, 5);
        std::cout <<R"( ____|_       ___|   |___.')";
        essential::gotoXY(73, 28);
        SetConsoleTextAttribute(h, 5);
        std::cout <<R"(/_/_____/____/_______|)";
    }
};

void selectMenu()
{
    int userChoice;
    addQuestionFile::addQuestionToFile addQues;
    gameHistory::StudentGameHistory sg;
    titleScreen();
    MainMenu menuAccess;
    menuAccess.dogDesign();

    menuAccess.mainMenu();
    std::cin >> userChoice;

    switch (userChoice)
    {
        case 1:
            menuAccess.selectMode();
            break;

        case 2:
            menuAccess.gameInstruction();
            break;

        case 3:
            sg.displayStudentData(&selectMenu);
            break;

        case 4:
            addQues.askTypeOfQues(&selectMenu);
```

```
            break;

        case 5:
            std::cout << "THANK YOU FOR PLAYING!";
            std::exit(0); //exit the program
            break;

        default:
            selectMenu(); // if the user input the wrong input the program wil ask again
            break;
    }
}

void titleScreen()
{
system("cls");
essential::gotoXY(29, 1);
SetConsoleTextAttribute(h, 1);
std::cout <<R"(_________       .__                 __     __.__                          .___)";
essential::gotoXY(29, 2);
SetConsoleTextAttribute(h, 1);
std::cout <<R"(\_____  \ __ __|__|_____________ / \   / \__|_____________________ _______  __| _/)";
essential::gotoXY(29, 3);
SetConsoleTextAttribute(h, 1);
std::cout <<R"( /  / \ \|  | \ \___   /\___   / \  \/\/  / \___   /\___   /\__ \\_ __ \/ __ | )";
essential::gotoXY(29, 4);
SetConsoleTextAttribute(h, 1);
std::cout <<R"(/  \_/. \ | /  |/   / /   /  \      /|  |/   / /   / / __ \|  | \/ /_/ |)";
essential::gotoXY(29, 5);
SetConsoleTextAttribute(h, 1);
std::cout <<R"(\_____\ \_/____/|__/_____ \/_____\  \__/\ / |__/_____ \/_____ \(____  /__|  \____ |)";
essential::gotoXY(29, 6);
SetConsoleTextAttribute(h, 1);
std::cout <<R"(      \__>          \/     \/      \/        \/     \/    \/         \/)";

}

int main()
{
    essential::loading();
    gameHistory::StudentGameHistory sg;

    MainMenu menu;
    essential::setcolor(essential::WHITE);
    system("cls");
    sg.getStudentInfo();
    selectMenu();
}

//TODO: this is what i should do first
//??Add error handling
//TODO: add limitation in quiz
        //if quiz is less than 5 then user can't load the file
        //Adding question in file if the use enter the same question in the file user can't add the question
//TODO: fix the position and color of each text
//TODO: finish the unfinish code in tsome of the mode
//TODO: refactor the code
//TODO: make the essential header more organize
```

```
//??
//TODO: Need to create an instruction

//TODO: Need to create an about section

//TODO: add a functionality in each mode in which the program will ask the user what type of question does the user want

//TODO: create feature in which the teacher / user can search for certain student and see result only for that student

//TODO: I need to create a feature in which the student can choose to see his result only and also see all of the result

//TODO:Finish all the design and of course the main menu
```

DESIGN

```
//Header guard
#ifndef ESSENTIAL_H
#define ESSENTIAL_H

#include <iostream>
#include <windows.h>
#include <stdio.h>
#include <random>
#include <vector>

namespace essential
{
   enum color
   { //enumeration is a user-defined data type
      NONE,
      DARK_BLUE,
      GREEN,
      DARK_CYAN,
      DARK_RED,
      PURPLE,
      DARK_YELLOW,
      NORMAL,
      GRAY,
      BLUE,
      LIME,
      CYAN,
      RED,
      PINK,
      YELLOW,
      WHITE
   };

   void ebod();
   void setcolor(color newColor);
   void gotoXY(int x, int y);
   void borderTwo();
   void ebodTwo();
   void border();
```

```
void asciiArtResult(void (*mainMenu)(void));
void askToMenu(void (*menu)(void));
void asciiArtGameOver(void (*func)(void));

void ebodTwo()
{

                for(int x = 20; x < 103; x++)
  {
    setcolor(DARK_CYAN);
    gotoXY(x,7);
    std::cout<<char(205);
                }

                for(int x = 8; x < 13; x++)
  {
                        setcolor(DARK_CYAN);
                        gotoXY(20,x);
                  std::cout<<char(186);
                }

                for(int x = 8; x < 13; x++)
  {
    setcolor(DARK_CYAN);
    gotoXY(103,x);
    std::cout<<char(186);
                }
                        setcolor(DARK_CYAN);

                gotoXY(20,7);
                std::cout<<char(201);
                gotoXY(103,7);
                std::cout<<char(187);
                gotoXY(103,13);
                std::cout<<char(188);
                gotoXY(20,13);
                std::cout<<char(200);
}

void ebod()
{

                for(int x = 20; x < 103; x++)
  {
    setcolor(DARK_CYAN);
    gotoXY(x,4);
    std::cout<<char(205);
                }

                for(int x = 20; x < 103; x++)
  {
                        setcolor(PURPLE);
                        gotoXY(x,13);
                        std::cout<<char(205);
                }

                for(int x = 5; x < 13; x++)
  {
```

```
                    setcolor(DARK_CYAN);
                    gotoXY(20,x);
                std::cout<<char(186);
                }

                for(int x = 5; x < 13; x++)
  {
    setcolor(DARK_CYAN);
    gotoXY(103,x);
    std::cout<<char(186);
                }
                    setcolor(DARK_CYAN);

                gotoXY(20,4);
                std::cout<<char(201);
                gotoXY(103,4);
                std::cout<<char(187);
                gotoXY(103,13);
                std::cout<<char(188);
                gotoXY(20,13);
                std::cout<<char(200);
}

void borderTwo()
{
  for(int x = 13; x < 16; x++)
  {
    setcolor(DARK_CYAN);
    gotoXY(20,x);
    std::cout<<char(186);
  }

  for(int x = 20; x < 103; x++)
  {
    setcolor(DARK_CYAN);
    gotoXY(x,16);
    std::cout<<char(205);
  }

  for(int x = 13; x < 16; x++)
  {
    setcolor(DARK_CYAN);
    gotoXY(103,x);
    std::cout<<char(186);
  }

  essential::setcolor(essential::DARK_CYAN);
  gotoXY(103,16);
  std::cout<<char(188);
  gotoXY(20,16);
  std::cout<<char(200);

}

void border()
{
  for(int x = 13; x < 16; x++)
  {
```

```
    setcolor(DARK_CYAN);
    gotoXY(20,x);
    std::cout<<char(186);
  }

  for(int x = 20; x < 103; x++)
  {
    setcolor(DARK_CYAN);
    gotoXY(x,16);
    std::cout<<char(205);
  }

  for(int x = 13; x < 16; x++)
  {
    setcolor(DARK_CYAN);
    gotoXY(103,x);
    std::cout<<char(186);
  }

  essential::setcolor(essential::DARK_CYAN);
  gotoXY(103,16);
  std::cout<<char(188);
  gotoXY(20,16);
  std::cout<<char(200);

}


void setcolor(color newColor)
{
  //A "handle" is a generic identifier (typically a pointer) used to represent something.
  SetConsoleTextAttribute( GetStdHandle(STD_OUTPUT_HANDLE), (newColor ) );
}

void gotoXY(int x, int y)  //function to decide location of the screem
{
  HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
  COORD CursorPosition;
  CursorPosition.X = x; // Locates column
  CursorPosition.Y = y; // Locates Row

  SetConsoleCursorPosition(console,CursorPosition); // Sets position for next thing to be printed
}

void loading()
{

  // Clear the console screen
  system("cls");

  // Enable VT100 terminal emulation to support special characters
  printf("\e[?251]");

  // Set ASCII to print special character.
  // Code page 437
  SetConsoleCP(437);
  SetConsoleOutputCP(437);
```

```
    // Define the characters for the loading bar
    int bar1 = 176, bar2 = 219;

    // Print the loading message
    gotoXY(50, 12);
    setcolor(BLUE);
    std::cout << "Loading. . . . .";

    // Print the empty loading bar
    gotoXY(50,  10);
    setcolor(BLUE);
    for(int i = 0; i < 25; i++)
        std::cout << (char)bar1;

    // Move the cursor back to the beginning of the loading bar
    std::cout << "\r";
    std::cout << "\t\t";

    // Print the filled loading bar
    gotoXY(50, 10);
    setcolor(BLUE);
    for(int i = 0; i < 25; i++)
    {
        std::cout << (char)bar2;
        // Wait for a short period of time to simulate the loading process
        Sleep(150);
    }
}

void asciiArtGameOver(void (*func)(void))
{
    ebodTwo();
    border();

    gotoXY(29,1);
    essential::setcolor(essential::RED);
    std::cout <<R"( _____                    _____            )";
    gotoXY(29,2);
    essential::setcolor(essential::RED);
    std::cout <<R"(|  __ \                  | _  |           )";
    gotoXY(29,3);
    essential::setcolor(essential::RED);
    std::cout <<R"(| |  \/ __ _ _ __ ___   ___  | | | |_  _____ _ __ )";
    gotoXY(29,4);
    essential::setcolor(essential::RED);
    std::cout <<R"(| | __ / _` | '_ ` _ \ / _ \ | | | \ \ / / _ \ '__|)";
    gotoXY(29,5);
    essential::setcolor(essential::RED);
    std::cout <<R"(| |_\ \ |_| | | | | | |  __/ \ \_/ /\ V /  __/ |)";
    gotoXY(29,6);
    essential::setcolor(essential::RED);
    std::cout <<R"(\____/\__,_|_| |_| |_|\___|  \___/  \_/ \___|_|)";

    askToMenu(func);
}

void askToMenu(void (*menu)(void)) //passing the function as parameter using pointers
{
```

```
    char choiceUser{};

    do
    {
       gotoXY(55, 17);
       std::cout << "Do you want to continue (y/n): ";
       std::cin >> choiceUser;

    }while(choiceUser != 'y' && choiceUser != 'n');

    if (choiceUser == 'y')
    {
       menu();
    }

    else
    {
       std::exit(0);
    }
  }

  void asciiArtResult(void (*mainMenu)(void))
  {
    ebodTwo();
    border();

    gotoXY(29,1);
    essential::setcolor(essential::GREEN);
    std::cout <<R"(_______ _____ _____ _   _ _   _____ )";
    gotoXY(29,2);
    essential::setcolor(essential::GREEN);
    std::cout <<R"(| ___ \ ___/  ___| | | | | |_   _|)";
    gotoXY(29,3);
    essential::setcolor(essential::GREEN);
    std::cout <<R"(| |_/ / |__ \ `--.| | | | |    | |)";
    gotoXY(29,4);
    essential::setcolor(essential::GREEN);
    std::cout <<R"(|    /|  __| `--. \ | | | |    | |)";
    gotoXY(29,5);
    essential::setcolor(essential::GREEN);
    std::cout <<R"(| |\ \| |___/\__/ / |_| | |____| |)";
    gotoXY(29,6);
    essential::setcolor(essential::GREEN);
    std::cout <<R"(\_| \_\____/\____/ \___/\_____/\_/)";

    askToMenu(mainMenu);
  }

  //this function is responsible for randoming the quetsion
  std::mt19937 randomingQuestion()
  {
    std::random_device rd;
          std::seed_seq ss{ rd(), rd(), rd(), rd(), rd(), rd(), rd(), rd() };
    std::mt19937 mt{ ss };

    return mt;
  }
}
```

```
#endif
```

USER GAME HISTORY

```
//Adding header guard
#ifndef USERGAMEHISTORY_H
#define USERGAMEHISTORY_H

#include <iostream>
#include <fstream>
#include <string>
#include "essential.h"
#include <vector>
#include <iomanip> // for setw function
#include <conio.h>
#include <cctype> //detect if the input is not number
#include <algorithm>

namespace gameHistory
{
   int g_studentNumber;
   std::string g_studentName;

   class StudentGameHistory
   {
   private:
      std::ofstream outFile;
      std::ifstream inFile;

   public:
      bool isTimeTrialMode { false };
      bool isSurvivalMode { false };
      bool isPassAndPlay { false };
      bool isClassicMode { false };
      bool hasEnterAMode  { false };

      std::string timeTrialMode { "Time Trial" };
      std::string survivalMode { "Survival" };
      std::string passAndPlayMode { "Pass & Play" };
      std::string classicMode { "Classic" };

      std::string gameMode{};
      std::string feedback{};

      //member functions
      void getStudentInfo();
      // void showStudentData();
```

```
    void writeStudentData(int score);
    void displayStudentData(void (*menu)(void));
    std::string  detectingWhatMode();
    double calculatePercent(int playerScore);
    std::string calculateGrade (int per);
    void openFilling();
    void showData(const std::vector <std::string>& v);
    std::string getFeedback(int score);

    bool openingFile()
    {
       if (!outFile.is_open())
       {
          std::cout << "ERROR: we can't open\n";
          return false;
       }

       else
       {
          return true;
       }
    }

    bool check_number(std::string& str)
    {
       for (int i = 0; i < str.length(); i++)
       {
          if (isdigit(str[i]) == false)
             return false;
             return true;
       }

    }

    //validate the name of the user
    bool isValidName(std::string& name)
    {
       return all_of(name.begin(), name.end(), [](char ch) {
          return (isalpha(ch) || isspace(ch));
       });
    }

    HANDLE h = GetStdHandle(STD_OUTPUT_HANDLE);
};

//ask the student information like name and student no.
```

```
void StudentGameHistory::getStudentInfo()
{
   essential::gotoXY(50, 5);
   std::cout << "Enter your student number: ";
   std::cin >> g_studentNumber;

   while ( !std::cin )
   {
      system("cls");
      std::cin.clear();
      std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
      essential::gotoXY(50, 5);
      std::cout << "Enter your student number: ";
      std::cin >> g_studentNumber;
   }

   do
   {
      essential::gotoXY(50, 7);
      std::cout << "Enter student name: ";
      //We use getline to be able to get all of the next character
      std::getline(std::cin >> std::ws, g_studentName); //using std::ws to ignore leading whitespaces
   }while(!isValidName(g_studentName));
}

std::string StudentGameHistory::detectingWhatMode()
{
   if ( isTimeTrialMode )
   {
      gameMode = timeTrialMode;
   }

   else if ( isSurvivalMode )
   {
      gameMode = survivalMode;
   }

   else if ( isPassAndPlay )
   {
      gameMode = passAndPlayMode;
   }

   else if ( isClassicMode )
   {
      gameMode = classicMode;
   }
```

```
    return gameMode;
}

void StudentGameHistory::writeStudentData(int score)
{
    outFile.open("studentRecord.txt", std::ios::out | std::ios::app);

    while ( !openingFile() )
    {
        openingFile();
    }

    outFile << g_studentNumber << std::endl;
    outFile << g_studentName << std::endl;
    outFile << score << std::endl;
    outFile << detectingWhatMode() << std::endl;
    outFile << calculatePercent(score) << std::endl;
    outFile << calculateGrade(score) << std::endl;
}

void StudentGameHistory::displayStudentData(void (*menu)(void))
{
    system("cls");

    SetConsoleTextAttribute(h, 6);
    std::cout << "ALL STUDENT RECORDS\n";
    SetConsoleTextAttribute(h, 2);
    std::cout<<
"=============================================================================
=\n";
    std::cout<<"StudentNo      Name          Score        quizMode          %          grade " << '\n';

std::cout<<"=============================================================================
================\n";

    openFilling();

    std::cout << "Press any key to menu. . . ";
    getch();
    system("cls");
    menu(); // go back to menu
}

double StudentGameHistory::calculatePercent(int playerScore)
{
    //calculating the percentage of the score of the student
```

```
    double percent = playerScore * 10.0 / 100.0;

    return percent;
}

std::string StudentGameHistory::calculateGrade (int per)
{
    if ( per >= 80 )
    {
        return "A";
    }

    else if ( per >= 60 )
    {
        return "B";
    }

    else if ( per >= 40 )
    {
        return "C";
    }

    else
    {
        return "F";
    }
}

std::string StudentGameHistory::getFeedback(int score)
{
    if ( score >= 80 )
    {
        return "Explendid!";
    }

    else if ( score >= 60 )
    {
        return "Great Job!";
    }

    else if ( score >= 40 )
    {
        return "Superb!";
    }

    else
```

```
        {
            return "Better Luck Next Time!";
        }
    }

    void StudentGameHistory::openFilling()
    {
        std::string filename("studentRecord.txt") ;
        std::ifstream in(filename.c_str()) ;

        if ( !in.is_open() )
        {
            std::cerr << "Unable to open " << filename << '\n' ;
        }

        std::vector <std::string> recordFiles;
        std::string records;

        //Filling the vector and using push back to allow that
        while ( std::getline(in, records) )
            recordFiles.push_back(records) ;

        showData(recordFiles);
    }

    void StudentGameHistory::showData(const std::vector <std::string>& v)
    {
         for ( unsigned i { 0 }; i < v.size(); ++i )
        {
            std::cout << v[i] << std::setw(20);

            //if the line reach 6 row then the program will add new line
            if ((i + 1) % 6 == 0)
            {
                std::cout << '\n';
            }
        }
    }
}

#endif
```

ADDING QUESTIONS

```
//add header guard
#ifndef ADDQUESTION_H
#define ADDQUESTION_H
```

```
#include <iostream>
#include <fstream>
#include <string>
#include <conio.h>

namespace addQuestionFile
{
   class addQuestionToFile
   {
   private:
      std::ofstream outFile;
      std::string fileName;
      std::string question;
      std::string option;
      char answer;
      int questionNumber{};
      int typeQues{};

      std::ofstream oFile;

   public:
      bool openTheFile();
      void askTypeOfQues(void (*func)(void));
      void addMultipleChoiceQues(void (*func)(void));
      void addIdentificationQues(void (*func)(void));

      HANDLE h = GetStdHandle(STD_OUTPUT_HANDLE);
   };

   bool addQuestionToFile::openTheFile()
   {
      system("cls");
      essential::gotoXY(50, 5);
      std::cout << "Add File Name: ";
      std::cin >> fileName;

      //increment the .txt
      fileName += ".txt";

      oFile.open(fileName.c_str(), std::ios::out | std::ios::app);

      if (!oFile.is_open())
      {
         std::cout << "ERROR: we can't open file:" << fileName << '\n';
         return false;
```

```
    }

    else
    {
        std::cout << "File " << oFile.is_open() << " is successfully open!\n";
        return true;
    }

}

void addQuestionToFile::askTypeOfQues(void (*func)(void))
{
    system("cls");
    essential::gotoXY(50, 5);
    SetConsoleTextAttribute(h, 15);
    std::cout << "-=-=-=-= Choose type of Question =-=-=-=-";
    essential::gotoXY(50, 7);
    SetConsoleTextAttribute(h, 15);
    std::cout << "1--> Multiple Choices Questions";
    essential::gotoXY(50, 9);
    SetConsoleTextAttribute(h, 15);
    std::cout << "2--> Identification";
    essential::gotoXY(50, 11);
    SetConsoleTextAttribute(h, 15);
    std::cout << "Choice: ";
    std::cin >> typeQues;

    switch(typeQues)
    {
        case 1:
            addMultipleChoiceQues(func);
            break;
        case 2:
            addIdentificationQues(func);
            break;
    }
}

void addQuestionToFile::addMultipleChoiceQues(void (*func)(void))
{
    openTheFile();
    system("cls");
    essential::gotoXY(50, 5);
    SetConsoleTextAttribute(h, 15);
    std::cout << "Number of questions: ";
    std::cin >> questionNumber;
```

```
while ( !std::cin || questionNumber < 6 )
{
   system("cls");
   std::cin.clear();
   std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
   essential::gotoXY(50, 5);
   std::cout << "Number of questions: ";
   std::cin >> questionNumber;
}

essential::gotoXY(50, 7);
SetConsoleTextAttribute(h, 15);
std::cout << "Consider the following format";
essential::gotoXY(50, 9);
SetConsoleTextAttribute(h, 1);
std::cout << "Question: ";
essential::gotoXY(50, 11);
SetConsoleTextAttribute(h, 7);
std::cout << "Choice1:";
essential::gotoXY(50, 12);
SetConsoleTextAttribute(h, 7);
std::cout << "Choice2: ";
essential::gotoXY(50, 13);
SetConsoleTextAttribute(h, 7);
std::cout << "Choice3: ";
essential::gotoXY(50, 14);
SetConsoleTextAttribute(h, 7);
std::cout << "Choice4: ";
essential::gotoXY(50, 16);
SetConsoleTextAttribute(h, 7);
std::cout << "Answer:";

for (int i { 0 }; i < questionNumber; ++i)
{
   SetConsoleTextAttribute(h, 7);
   essential::gotoXY(50, 17);
   std::cout << "Press any key to next";
   getch();
   system("cls");
   SetConsoleTextAttribute(h, 15);
   std::cout << "Please Enter The Question and Answer: \n";
   SetConsoleTextAttribute(h, 15);
   std::cout << "Queston #" << i + 1 << '\n';
   std::cin.get();
   std::getline(std::cin >> std::ws, question);
```

```
        oFile << question << std::endl;

        for (int x { 0 }; x < 4; ++x)
        {
            std::cin.get();
            std::getline(std::cin >> std::ws, option);
            oFile << option << std::endl;
        }

        std::cin >> answer;
        oFile << answer << std::endl;
    }

    essential::askToMenu(func);
    oFile.close();
}

void addQuestionToFile::addIdentificationQues(void (*func)(void))
{
    openTheFile();

    system("cls");
    std::cout << "Number of questions: ";
    std::cin >> questionNumber;

    while ( !std::cin || questionNumber < 6 )
    {
        system("cls");
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        essential::gotoXY(50, 5);
        SetConsoleTextAttribute(h, 15);
        std::cout << "Number of questions: ";
        std::cin >> questionNumber;
    }

    essential::gotoXY(50, 5);
    SetConsoleTextAttribute(h, 15);
    std::cout << "Consider the following format" << '\n';
    essential::gotoXY(50, 7);
    SetConsoleTextAttribute(h, 1);
    std::cout << "\nQuestion: \n";
    essential::gotoXY(50, 9);
    SetConsoleTextAttribute(h, 7);
    std::cout << "\nAnswer:\n";
```

```
        for (int i { 0 }; i < questionNumber; ++i)
        {
            SetConsoleTextAttribute(h, 1);
            essential::gotoXY(50, 10);
            std::cout << "Press any key to next";
            getch();
            system("cls");
            SetConsoleTextAttribute(h, 15);
            std::cout << "Please Enter The Question and Choices: \n";
            SetConsoleTextAttribute(h, 15);
            std::cout << "Queston #" << i + 1 << '\n';
            std::cin.get();
            std::getline(std::cin >> std::ws, question);
            oFile << question << std::endl;
            //getting the answer
            std::cin >> answer;
            oFile << answer << std::endl;
        }

        essential::askToMenu(func);

        oFile.close();
    }
}

#endif
```

QUESTION IDENTIFICATION

```
//Header Guard
#ifndef QUESTIONIDENTIFICATION_H
#define QUESTIONIDENTIFICATION_H

#include <vector>
#include <fstream>
#include <iostream>
#include <string>
#include "essential.h"
#include "userGameHistory.h"

namespace questionIdentification
{
    int g_scoreOfPlayer{};

    class Question
    {
    private:
```

```
    std::string questions;
    std::string answer;

public:
    Question(std::string& ques, std::string& ans);

    std::string getQuestion(){return questions;};
    std::string getAnswer(){return answer;};
};

Question::Question(std::string& ques, std::string& ans)
{
    questions = ques;
    answer = ans;
}

class gettingQuesFile
{
public:
    int questionNum;
    int inCorrectAns;

    gameHistory::StudentGameHistory sg;

    bool openFile(std::ifstream& quesFile)
    {
        std::string fileName;

        system("cls");
        essential::gotoXY(23, 7);
        std::cout << "How many questions does file haves: ";
        std::cin >> questionNum;

        //detect if user input is an integer or not and also validate if user input the right question dami
        while ( !std::cin || questionNum < 6 )
        {
            system("cls");
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            essential::gotoXY(23, 7);
            std::cout << "How many questions does file haves: ";
            std::cin >> questionNum;
        }

        do
        {
```

```
        system("cls");
        essential::gotoXY(23, 5);
        std::cout << "Add .txt at the end of your filename.";
        essential::gotoXY(23, 6);
        std::cout << "Filename: ";
        std::getline(std::cin >> std::ws, fileName);

        quesFile.open(fileName);

    }while( !quesFile.is_open() ); //this detect if the user input the write file name or not

    if (!quesFile.is_open())
    {
        essential::gotoXY(23, 6);
        Sleep(1000);
        std::cout << "Could not find database. . . " << '\n';
        return true;
    }

    else
    {
        essential::gotoXY(23, 6);
        Sleep(1000);
        std::cout << "Database sucessfully accessed!" << '\n';
        return false;
    }
}

void fillVector(std::vector <Question>& newQuestionList, std::ifstream& queFile)
{
    std::string questions;
    std::string answer;

    for (int i = 0; i < questionNum; i++)
    {
        std::getline(queFile >> std::ws, questions);
        std::getline(queFile >> std::ws, answer);

        Question newQuestion(questions, answer);
        newQuestionList.push_back(newQuestion);

        std::shuffle(newQuestionList.begin(), newQuestionList.end(), essential::randomingQuestion());
    }
}

void printVector(std::vector <Question>& newQuestionList, void (*menu)(void))
```

```
{
    std::string choice;
    system("cls");
    int i;

    for(i = 0; i < questionNum; i++)
    {
        essential::gotoXY(55, 18);
        std::cout << "Press any key to next. . . .";
        essential::gotoXY(55, 17);
        getch();
        system("cls");

        essential::ebod();
        essential::borderTwo();

        essential::gotoXY(23, 5);
        SetConsoleTextAttribute(h, 15);
        std::cout << "Question # " << i + 1;
        essential::gotoXY(23, 8);
        SetConsoleTextAttribute(h, 1);
        std::cout <<  newQuestionList[i].getQuestion();
        essential::gotoXY(23, 14);
        SetConsoleTextAttribute(h, 7);
        std::cout << "Fill Answer: ";
        std::getline(std::cin >> std::ws, choice);

        if (choice == newQuestionList[i].getAnswer())
        {
            essential::gotoXY(55, 17);
            SetConsoleTextAttribute(h, 2);
            std::cout << "Congratulation You selected right option";
            g_scoreOfPlayer++;
        }

        else
        {
            essential::gotoXY(55, 17);
            SetConsoleTextAttribute(h, 4);
            std::cout << "You selected wrong option.";
        }
    }

    if (i == questionNum)
    {
        system("cls");
```

```
            sg.writeStudentData(g_scoreOfPlayer);
            resultScreen(menu);
        }
    }

    void start(void (*menu)(void))
    {
        std::ifstream questionFile;
        std::vector<Question> questionList;

        sg.isClassicMode = true;
        //If openfile return true it means the user input is wrong if return false then it was write

        openFile(questionFile);

        fillVector(questionList, questionFile);
        printVector(questionList, menu);
    }

    void resultScreen(void (*func)(void))
    {
        essential::gotoXY(35, 10);
        SetConsoleTextAttribute(h, 15);
        std::cout << "Student Name: " << gameHistory::g_studentName;
        essential::gotoXY(70, 10);
        SetConsoleTextAttribute(h, 15);
        std::cout << "Student No. " << gameHistory::g_studentNumber;
        essential::gotoXY(35, 13);
        SetConsoleTextAttribute(h, 15);
        std::cout << "Your score ---> " << g_scoreOfPlayer;
        essential::gotoXY(70, 12);
        SetConsoleTextAttribute(h, 15);
        std::cout << sg.getFeedback(g_scoreOfPlayer);
        essential::gotoXY(70, 14);
        SetConsoleTextAttribute(h, 15);
        std::cout << inCorrectAns;

        //call result ascii art and also the border
        essential::asciiArtResult(func);
    }

    HANDLE h = GetStdHandle(STD_OUTPUT_HANDLE);
  };
}

#endif
```

MULTIPLE CHOICE

```
//Header Guard

#ifndef QUESTIONMCQ_H
#define QUESTIONMCQ_H

#include <vector>
#include <fstream>
#include <iostream>
#include <string>
#include <windows.h>
#include <conio.h>
#include "essential.h"
#include <random>
#include <cctype>
#include "userGameHistory.h"

namespace questionMCQ
{
   int g_scoreOfPlayer{};

   class Question
   {
   private:
      std::string questions;
      std::string optionOne;
      std::string optionTwo;
      std::string optionThree;
      std::string optionFour;
      char answer;

   public:
      Question(std::string& ques, std::string& opOne, std::string& opTwo, std::string& opThree, std::string&
opFour, char ans);

      std::string getQuestion(){return questions;};
      std::string getOptionOne(){return optionOne;};
      std::string getOptionTwo(){return optionTwo;};
      std::string getOptionThree(){return optionThree;};
      std::string getOptionFour(){return optionFour;};
      char getAnswer(){return answer;};
   };

   Question::Question(std::string& ques, std::string& opOne, std::string& opTwo, std::string& opThree,
std::string& opFour, char ans)
   {
```

```
    questions = ques;
    optionOne = opOne;
    optionTwo = opTwo;
    optionThree = opThree;
    optionFour = opFour;
    answer = ans;
}

class fileDoing
{
public:
    int askUserQues{};
    int playerScore{};
    int isTrue {};

    gameHistory::StudentGameHistory sg;

    bool openFile(std::ifstream& quesFile)
    {
        std::string fileName;

        system("cls");
        essential::gotoXY(23, 7);
        SetConsoleTextAttribute(h, 15);
        std::cout << "How many questions does file have: ";
        std::cin >> askUserQues;

        //detect if user input is an integer or not and also validate if user input the right question dami
        while ( !std::cin || askUserQues < 6 )
        {
            system("cls");
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            essential::gotoXY(23, 7);
            SetConsoleTextAttribute(h, 15);
            std::cout << "How many questions does file have: ";
            std::cin >> askUserQues;
        }

        do
        {
            system("cls");
            essential::gotoXY(23, 5);
            SetConsoleTextAttribute(h, 15);
            std::cout << "Add .txt at the end of your filename.";
            essential::gotoXY(23, 6);
```

```
        std::cout << "Filename: ";
        std::getline(std::cin >> std::ws, fileName);

        quesFile.open(fileName);

    }while( !quesFile.is_open() ); //this detect if the user input the write file name or not

    if (!quesFile.is_open())
    {
        essential::gotoXY(23, 6);
        Sleep(1000);
        SetConsoleTextAttribute(h, 15);
        std::cout << "Could not find database. . . " << '\n';
        return true;
    }

    else
    {
        essential::gotoXY(23, 6);
        Sleep(1000);
        SetConsoleTextAttribute(h, 15);
        std::cout << "Database sucessfully accessed!" << '\n';
        return false;
    }
}

void fillVector(std::vector <Question>& newQuestionList, std::ifstream& queFile)
{
    std::string questions;
    std::string optionOne;
    std::string optionTwo;
    std::string optionThree;
    std::string optionFour;
    char answer;

    for (int i = 0; i < askUserQues; i++)
    {
        std::getline(queFile >> std::ws, questions);
        std::getline(queFile >> std::ws, optionOne);
        std::getline(queFile >> std::ws, optionTwo);
        std::getline(queFile >> std::ws, optionThree);
        std::getline(queFile >> std::ws, optionFour);
        queFile >> answer;
        queFile.ignore(1000, '\n');

        Question newQuestion(questions, optionOne, optionTwo, optionThree, optionFour, answer);
```

```
        newQuestionList.push_back(newQuestion);

        //random questions
        std::shuffle(newQuestionList.begin(), newQuestionList.end(), essential::randomingQuestion());
    }
}

void printVectorOne(std::vector <Question>& newQuestionList, void (*menu)(void))
{
    char choice;

    int questionNumber{};
    int i{};
    char a;

    do
    {
        if ( questionNumber == i )
        {
            system("cls");

            questionNumber++;

            generateQuestion(questionNumber, i, newQuestionList, true);
        }

        if ( _kbhit() )
        {
            a = _getch();

            // essential::gotoXY(56, 17);
            // std::cout << a;

            if(int(a)==13)
            {
                essential::gotoXY(55, 17);
                SetConsoleTextAttribute(h, 7);
                std::cout << "You skipped this Question";
            }

            else
            {
                if( a == newQuestionList[i].getAnswer())
                {
                    essential::gotoXY(55, 17);
                    SetConsoleTextAttribute(h, 4);
```

```
                std::cout << "Congratulation You selected right option";
                g_scoreOfPlayer++;
            }

            else
            {
                essential::gotoXY(55, 17);
                SetConsoleTextAttribute(h, 2);
                std::cout << "You selected wrong option.";
            }
        }
        _getch();
        i++;
    }
  }while( i < askUserQues );

  if ( i == askUserQues )
  {
    system("cls");
    sg.writeStudentData(g_scoreOfPlayer);
    resultScreen(menu);
    system("cls");
  }
}

void start(void (*menu)(void))
{
  std::ifstream questionFile;
  std::vector <Question> questionList; //vector to tore the text from the file
  sg.isClassicMode = true;
  //If openfile return true it means the user input is wrong if return false then it was write
  openFile(questionFile);

  fillVector(questionList, questionFile);
  printVectorOne(questionList, menu);
}

void generateQuestion(int ques, int in, std::vector <Question>& v, bool isTrue)
{
  system("cls");

  essential::ebod();
  essential::borderTwo();

  SetConsoleTextAttribute(h, 15);
  essential::gotoXY(23, 5);
```

```
    std::cout << "Question # " << ques << '\n';
    SetConsoleTextAttribute(h, 1);
    essential::gotoXY(23, 7);
    std::cout << v[in].getQuestion();
    SetConsoleTextAttribute(h, 7);
    essential::gotoXY(23, 9);
    std::cout << "A. " << v[in].getOptionOne() << '\n';
    essential::gotoXY(23, 10);
     SetConsoleTextAttribute(h, 7);
    std::cout << "B. " << v[in].getOptionTwo() << '\n';
    SetConsoleTextAttribute(h, 7);
    essential::gotoXY(23, 11);
    std::cout << "C. " << v[in].getOptionThree() << '\n';
    SetConsoleTextAttribute(h, 7);
    essential::gotoXY(23, 12);
    std::cout << "D. " <<v[in].getOptionFour() << '\n';
    SetConsoleTextAttribute(h, 14);
    essential::gotoXY(23, 14);
    std::cout << "Select choices between (a - d): ";

    if (isTrue)
    {
        essential::color(essential::YELLOW);
        SetConsoleTextAttribute(h, 14);
        essential::gotoXY(23, 15);
        std::cout << "Press enter to skip";
    }

    else
    {
      //empty
    }
}

HANDLE h = GetStdHandle(STD_OUTPUT_HANDLE);

void resultScreen(void (*func)(void))
{
    essential::gotoXY(35, 10);
    SetConsoleTextAttribute(h, 15);
    std::cout << "Student Name: " << gameHistory::g_studentName;
    essential::gotoXY(70, 10);
    SetConsoleTextAttribute(h, 15);
    std::cout << "Student No. " << gameHistory::g_studentNumber;
    essential::gotoXY(35, 13);
    SetConsoleTextAttribute(h, 15);
```

```
            std::cout << "Your score ---> " << g_scoreOfPlayer;
            essential::gotoXY(70, 12);
            SetConsoleTextAttribute(h, 15);
            std::cout << sg.getFeedback(g_scoreOfPlayer);
            essential::gotoXY(70, 14);

            //call result ascii art and also the border
            essential::asciiArtResult(func);
        }
    };
}


//This i s a header guard
#ifndef CLASSICMODE_H
#define CLASSICMODE_H

#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <cstdlib>
#include "questionMultipleChoice.h"
#include "questionIdentification.h"
#include <fstream>

// Adding namespace to avoid naming collision
namespace classicMode
{
    class ClassicMode
    {
    private:
        int userChoice{};
        std::string userInputtedAns{};

        std::ifstream questionFile;
        std::vector <questionMCQ::Question> questionListVect;
        char yAndNChoice{};

        questionMCQ::fileDoing gettingFiles;
        questionIdentification::gettingQuesFile fileGathering;

    public:
        int playerScore{};
        void startGame(void (*mainMenuAccess)(void));
```

```
    void askTypeOfQuestion(void (*menu)(void))
    {
       int userChoice{};

       system("cls");
       essential::gotoXY(50, 5);
       SetConsoleTextAttribute(h, 15);
       std::cout << "-=-=- Select type of question -=-=-";
       essential::gotoXY(50, 7);
       SetConsoleTextAttribute(h, 15);
       std::cout << "[1] Multiple Choices Question";
       essential::gotoXY(50, 9);
       SetConsoleTextAttribute(h, 15);
       std::cout << "[2] Identification";
       essential::gotoXY(50, 11);
       SetConsoleTextAttribute(h, 15);
       std::cout << "Select: ";
       std::cin >> userChoice;

       switch(userChoice)
       {
          case 1:
             gettingFiles.start(menu);
             playerScore = questionMCQ::g_scoreOfPlayer;
             break;

          case 2:
             fileGathering.start(menu);
             playerScore = questionIdentification::g_scoreOfPlayer;
             break;
       }
    }

    HANDLE h = GetStdHandle(STD_OUTPUT_HANDLE);
  };

  void ClassicMode::startGame(void (*mainMenuAccess)(void))
  {
      askTypeOfQuestion(mainMenuAccess);
  }
}

#endif

#endif
```

CLASSIC MODE

```
//This i s a header guard
#ifndef CLASSICMODE_H
#define CLASSICMODE_H

#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <cstdlib>
#include "questionMultipleChoice.h"
#include "questionIdentification.h"
#include <fstream>

// Adding namespace to avoid naming collision
namespace classicMode
{
   class ClassicMode
   {
   private:
      int userChoice{};
      std::string userInputtedAns{};

      std::ifstream questionFile;
      std::vector <questionMCQ::Question> questionListVect;
      char yAndNChoice{};

      questionMCQ::fileDoing gettingFiles;
      questionIdentification::gettingQuesFile fileGathering;

   public:
      int playerScore{};
      void startGame(void (*mainMenuAccess)(void));

      void askTypeOfQuestion(void (*menu)(void))
      {
         int userChoice{};

         system("cls");
         essential::gotoXY(50, 5);
         SetConsoleTextAttribute(h, 15);
         std::cout << "-=-=- Select type of question -=-=-";
         essential::gotoXY(50, 7);
         SetConsoleTextAttribute(h, 15);
         std::cout << "[1] Multiple Choices Question";
```

```
        essential::gotoXY(50, 9);
        SetConsoleTextAttribute(h, 15);
        std::cout << "[2] Identification";
        essential::gotoXY(50, 11);
        SetConsoleTextAttribute(h, 15);
        std::cout << "Select: ";
        std::cin >> userChoice;

        switch(userChoice)
        {
           case 1:
              gettingFiles.start(menu);
              playerScore = questionMCQ::g_scoreOfPlayer;
              break;

           case 2:
              fileGathering.start(menu);
              playerScore = questionIdentification::g_scoreOfPlayer;
              break;
        }
     }

     HANDLE h = GetStdHandle(STD_OUTPUT_HANDLE);
  };

  void ClassicMode::startGame(void (*mainMenuAccess)(void))
  {
      askTypeOfQuestion(mainMenuAccess);
  }
}

#endif
```

PASS AND PLAY

```
//Header guard
#ifndef PASSANDPLAY_H
#define PASSANDPLAY_H

#include <iostream>
#include <vector>
#include <algorithm>
#include <string> //To use getline function
#include <windows.h> // for sleep function
#include <conio.h>
```

```
#include "essential.h"
#include "questionMultipleChoice.h"
#include "userGameHistory.h"

#define TOTAL_SCORE 100

namespace passAndPlay
{
  class PassAndPlay
   {
   private:

      std::vector <questionMCQ::Question> listOfQuestionPOne;
      std::vector <questionMCQ::Question> listOfQuestionPTwo;

      questionMCQ::fileDoing getFile;
      gameHistory::StudentGameHistory sg;

      int playerOneScore{};
      int playerTwoScore{};

      int quesNum { 0 };
      int index { 0 };
      char a{};
      int numQuestions{};
      std::string whoWin{};

      bool makeTurn = true;

      std::ifstream questionFileListOne;
      std::ifstream questionFileListTwo;

   public:
    //This is where variables should go
      std::string playerNameOne{};
      std::string playerNameTwo{};

      void askPlayerName();
      void makeTurns(void (*pass)(void));
      void performAskingUser(std::string&, void (*access)(void));
      void startGame(void (*passMenu)(void));
      void questionsTwo(std::string& pNameTwo, void (*accessReqTwo)(void));
      void questionsOne(std::string& pNameOne, void (*accessReqOne)(void));
      void resultScreen(void (*func)(void), std::string&);

      void displayQuestion(std::string& playerNames, std::vector <questionMCQ::Question> listOfQuestion,
void (*menuAccess)(void))
```

```
        {
            auto itOne = listOfQuestionPOne.begin();
            auto itTwo = listOfQuestionPTwo.begin();

            while (true)
            {
               for (int x { 0 }; x < getFile.askUserQues; ++x)
               {
                  Sleep(1000);
                  system("cls");

                  essential::ebod();
                  essential::borderTwo();

                  essential::gotoXY(23, 5);
                  std::cout << "Question # " << x + 1<< '\n';
                  essential::gotoXY(23, 7);
                  std::cout << listOfQuestion[x].getQuestion();
                  essential::gotoXY(23, 9);
                  std::cout << "A. " << listOfQuestion[x].getOptionOne() << '\n';
                  essential::gotoXY(23, 10);
                  std::cout << "B. " << listOfQuestion[x].getOptionTwo() << '\n';
                  essential::gotoXY(23, 11);
                  std::cout << "C. " << listOfQuestion[x].getOptionThree() << '\n';
                  essential::gotoXY(23, 12);
                  std::cout << "D. " << listOfQuestion[x].getOptionFour() << '\n';
                  essential::gotoXY(23, 14);
                  std::cout << "Press enter to skip";
                  essential::gotoXY(23, 15);
                  std::cout << "Select choices between (a - d) ";
                  std::cin >> a;

                  if( a == listOfQuestion[x].getAnswer())
                  {
                     essential::gotoXY(55, 17);
                     SetConsoleTextAttribute(h, 2);
                     std::cout << "Congratulation You selected right option";

                     //this increment the score of the two player depending who are playing
                     (playerNames == playerNameOne) ? playerOneScore += 10 : playerTwoScore += 10;

                     //this line of code erase the question of the player who get the correct answer
                     (playerNames == playerNameOne) ?  listOfQuestionPOne.erase(itOne) :
listOfQuestionPTwo.erase(itTwo);
                  }

                  else
```

```
{
   essential::gotoXY(55, 17);
   SetConsoleTextAttribute(h, 4);
   std::cout << "You selected wrong option.";

   if (playerNames == playerNameOne)
   {
      if (makeTurn)
      {
          makeTurn = false;
          makeTurns(menuAccess);
      }

      else
      {
         makeTurn = true;
         makeTurns(menuAccess);
      }
   }

   else
   {
      if (makeTurn)
      {
         makeTurn = false;
         makeTurns(menuAccess);
      }

      else
      {
         makeTurn = true;
         makeTurns(menuAccess);
      }
   }
}

if (playerOneScore == TOTAL_SCORE)
{
   system("cls");
   whoWin = "Congrats! Player One you win!!";
   resultScreen(menuAccess, whoWin);
   sg.writeStudentData(playerOneScore);
   std::exit(0);
}

else if (playerTwoScore == TOTAL_SCORE)
```

```
            {
               system("cls");
               whoWin = "Congrats! Player Two you win!!";
               resultScreen(menuAccess, whoWin);
               sg.writeStudentData(playerTwoScore);
               std::exit(0);
            }

            if ( x == getFile.askUserQues )
            {
               if ( playerOneScore > playerTwoScore )
               {
                  whoWin = "Congrats! Player one you win!!";
                  resultScreen(menuAccess, whoWin);
                  sg.writeStudentData(playerTwoScore);
                  std::exit(0);
               }

               else
               {
                  whoWin = "Congrats! Player Two you win!!";
                  resultScreen(menuAccess, whoWin);
                  sg.writeStudentData(playerTwoScore);
                  std::exit(0);
               }
            }
         }

         if (playerOneScore == TOTAL_SCORE)
         {
            sg.writeStudentData(playerOneScore);
            break;
         }

         else if (playerTwoScore == TOTAL_SCORE)
         {
            sg.writeStudentData(playerTwoScore);
            break;
         }
      }
   }

   HANDLE h = GetStdHandle(STD_OUTPUT_HANDLE);
};

void PassAndPlay::startGame(void (*passMenu)(void))
```

```
{
   system("cls");
   sg.isPassAndPlay = true;
   askPlayerName();
   makeTurns(passMenu);
}

//Ask player name
void PassAndPlay::askPlayerName()
{
   essential::gotoXY(23, 10);
   SetConsoleTextAttribute(h, 1);
   std::cout << "Player one name: ";
   std::getline(std::cin >> std::ws, playerNameOne); //Using std::ws to ignore leading whitespaces
   essential::gotoXY(23, 12);
   SetConsoleTextAttribute(h, 9);
   std::cout << "Player two name: ";
   std::getline(std::cin >> std::ws, playerNameTwo);

   system("cls");
   essential::gotoXY(23, 5);
   SetConsoleTextAttribute(h, 15);
   std::cout << playerNameOne << " Input your question. . . . .  ";
   Sleep(2000);
   getFile.openFile(questionFileListOne);

   system("cls");
   essential::gotoXY(23, 5);
   SetConsoleTextAttribute(h, 15);
   std::cout << playerNameTwo << " Input your question. . . . .  ";
   Sleep(2000);
   getFile.openFile(questionFileListTwo);
}

//this function allow us to maketurn for the players
void PassAndPlay::makeTurns(void (*menuAccess)(void))
{
   if (makeTurn)
   {
      system("cls");
      performAskingUser(playerNameOne, menuAccess);
   }

   else
   {
      system("cls");
```

```
        performAskingUser(playerNameTwo, menuAccess);
    }
}

void PassAndPlay::performAskingUser(std::string& turnPlayerName, void (*access)(void))
{
    std::string userChoice{};

    if (turnPlayerName == playerNameOne)
    {
        while (true)
        {
            essential::gotoXY(23, 5);
            SetConsoleTextAttribute(h, 14);
            std::cout << turnPlayerName << " it's your turn. . . . . ";
            questionsOne(turnPlayerName, access);
            break;
        }
    }

    else
    {
         while (true)
        {
            essential::gotoXY(23, 5);
            SetConsoleTextAttribute(h, 14);
            std::cout << turnPlayerName << " it's your turn. . . . . ";
            questionsTwo(turnPlayerName, access);
            break;
        }
    }
}

//This function hold the questions for player two
void PassAndPlay::questionsTwo(std::string& pNameTwo, void (*accessReqTwo)(void))
{
    getFile.fillVector(listOfQuestionPTwo, questionFileListTwo);
    displayQuestion(pNameTwo, listOfQuestionPTwo, accessReqTwo);
}

//this function add question for the player one
void PassAndPlay::questionsOne(std::string& pNameOne, void (*accessReqOne)(void))
{
    getFile.fillVector(listOfQuestionPOne, questionFileListOne);
    displayQuestion(pNameOne, listOfQuestionPOne, accessReqOne);
}
```

```
    void PassAndPlay::resultScreen(void (*func)(void), std::string& str)
    {
        essential::gotoXY(35, 10);
        SetConsoleTextAttribute(h, 15);
        std::cout << "Student Name: " << gameHistory::g_studentName;
        essential::gotoXY(70, 10);
        SetConsoleTextAttribute(h, 15);
        std::cout << "Student No. " << gameHistory::g_studentNumber;
        essential::gotoXY(35, 13);
        SetConsoleTextAttribute(h, 15);
        std::cout << str;

        //call result ascii art and also the border
        essential::asciiArtResult(func);
    }
}

#endif
```

SURVIVAL

```
//Header guards
#ifndef SURVIVALMODE_H
#define SURVIVALMODE_H

#include <iostream>
#include <chrono>
#include <ctime>
#include <conio.h>
#include <windows.h> //for essential::gotoXY function
#include <vector>
#include <fstream> //for file handling
#include <string>
#include "questionMultipleChoice.h"
#include "userGameHistory.h"

namespace survivalModes
{
    class SurvivalMode
    {
    public:
        std::ifstream questionFile;
        std::vector<questionMCQ::Question> questionList;
```

```
questionMCQ::fileDoing getFile;

gameHistory::StudentGameHistory sg;

int quesNum { 0 };
int index { 0 };
char a{};
int playerScore{};
int isGameOver = true;
int life = 4;

std::vector <std::string> lives = {"X", "X", "X", "X"};

void gameOVerScreen(void (*func)(void));
void resultScreen(void (*func)(void));

void startQuiz(void (*funcs)(void))
{
   start();

   sg.isSurvivalMode = true;

   do
   {
      if (quesNum == index)
      {
         system("cls");

         quesNum++;

        essential::ebod();
         essential::borderTwo();

         essential::gotoXY(69, 3);
         SetConsoleTextAttribute(h, 5);
         //This output the reaming lives of the player
         std::cout << "Lives: " << lives[0] << lives[1] << lives[2] << lives[3];

         essential::gotoXY(23, 5);
         SetConsoleTextAttribute(h, 15);
         std::cout << "Question # " << quesNum << '\n';
         essential::gotoXY(23, 7);
         SetConsoleTextAttribute(h, 1);
         std::cout << questionList[index].getQuestion();
         essential::gotoXY(23, 9);
         SetConsoleTextAttribute(h, 7);
```

```
            std::cout << "A. " << questionList[index].getOptionOne() << '\n';
            essential::gotoXY(23, 10);
            SetConsoleTextAttribute(h, 7);
            std::cout << "B. " << questionList[index].getOptionTwo() << '\n';
            essential::gotoXY(23, 11);
            SetConsoleTextAttribute(h, 7);
            std::cout << "C. " << questionList[index].getOptionThree() << '\n';
            essential::gotoXY(23, 12);
            SetConsoleTextAttribute(h, 7);
            std::cout << "D. " <<questionList[index].getOptionFour() << '\n';
            essential::gotoXY(23, 14);
            SetConsoleTextAttribute(h, 14);
            std::cout << "Select choices between (a -d): ";
        }

        //kbhit function detect if the user input something
        if ( _kbhit() )
        {
            a = _getch();

            essential::gotoXY(56, 14);
            std::cout << a;

            if( a == questionList[index].getAnswer())
            {
                essential::gotoXY(55, 17);
                SetConsoleTextAttribute(h, 2);
                std::cout << "Congratulation You selected right option";
                playerScore += 10; //This will increment if the player get the correct answer
            }

            else
            {
                essential::gotoXY(55, 17);
                SetConsoleTextAttribute(h, 4);
                std::cout << "You selected wrong option.";
                lives[index].erase(); //if the player choose the wrong answer the lives will be erase
                --life;
            }

            _getch();
            index++;
        }
    }while(life != 0 && index < getFile.askUserQues); //if the remainingtime == 0 and the index reach the numeber of question that the user want the program will stop

    if (life == 0)
```

```
        {
            system("cls");
            sg.writeStudentData(playerScore);
            gameOVerScreen(funcs);

        }

        else if (index == getFile.askUserQues)
        {
            system("cls");
            sg.writeStudentData(playerScore);
            resultScreen(funcs);
        }
    }

    void start()
    {
        getFile.openFile(questionFile);

        getFile.fillVector(questionList, questionFile);
    }

    HANDLE h = GetStdHandle(STD_OUTPUT_HANDLE);
};

void SurvivalMode::gameOVerScreen(void (*func)(void))
{
    essential::gotoXY(35, 10);
    SetConsoleTextAttribute(h, 15);
    std::cout << "Student Name: " << gameHistory::g_studentName;
    essential::gotoXY(70, 10);
    SetConsoleTextAttribute(h, 15);
    std::cout << "Student No. " << gameHistory::g_studentNumber;
    essential::gotoXY(35, 13);
    SetConsoleTextAttribute(h, 15);
    std::cout << "Your score ---> " << playerScore;
    essential::gotoXY(70, 13);
    SetConsoleTextAttribute(h, 15);
    std::cout << sg.getFeedback(playerScore);

    //call game over ascii art and also the border
    essential::asciiArtGameOver(func);
}

void SurvivalMode::resultScreen(void (*func)(void))
{
```

```
        essential::gotoXY(35, 10);
        SetConsoleTextAttribute(h, 15);
        std::cout << "Student Name: " << gameHistory::g_studentName;
        essential::gotoXY(70, 10);
        SetConsoleTextAttribute(h, 15);
        std::cout << "Student No. " << gameHistory::g_studentNumber;
        essential::gotoXY(35, 13);
        SetConsoleTextAttribute(h, 15);
        std::cout << "Your score ---> " << playerScore;
        essential::gotoXY(70, 12);
        SetConsoleTextAttribute(h, 15);
        std::cout << sg.getFeedback(playerScore);
        essential::gotoXY(70, 14);
        SetConsoleTextAttribute(h, 15);
        std::cout << "Remaining Life: " << life;

        //call result ascii art and also the border
        essential::asciiArtResult(func);
    }
}

#endif
```

TIME TRIAL

```
//Header guards
#ifndef TIMETRIALMODE_H
#define TIMETRIALMODE_H

#include <iostream>
#include <chrono>
#include <ctime>
#include <conio.h>
#include <windows.h> //for essential::gotoXY function
#include <vector>
#include <fstream> //for file handling
#include <string>
#include "questionMultipleChoice.h"
#include "essential.h"
#include <random>
#include "userGameHistory.h"

namespace timeTrialModes
{
    class TimeTrialMode
```

```
{
public:
    std::ifstream questionFile;
    std::vector<questionMCQ::Question> questionList;

    questionMCQ::fileDoing gettingFile;
    gameHistory::StudentGameHistory sg;

    double maxtime { 65 };
    int quesNum { 0 };
    int index { 0 };
    char a;
    int userScore{};
    std::string feedBackLose{};
    int wrongAnswers{};

    void gameOverScreen(void (*func)(void));
    void resultScreen(void (*func)(void));

    void startQuiz(void (*menuAccessed)(void))
    {
        system("cls");
        time_t startTime, currentTime;
        int remainingTime;

        sg.isTimeTrialMode = true;

        startTime = time(NULL);

        getQuestion();

        do
        {
            currentTime = time(NULL);
            remainingTime = maxtime - difftime(currentTime, startTime);

            essential::gotoXY(69, 3);
            //output the reaming time
            SetConsoleTextAttribute(h, 5);
            std::cout << "Time remaining: " << remainingTime << " seconds\r" << std::flush;
            Sleep(1);

            if (quesNum == index)
            {
                quesNum++;
```

```
        //generating the question
        gettingFile.generateQuestion(quesNum, index, questionList, true);
    }

    //kbhit function detect if the user input something
    if ( _kbhit() )
    {
        a = _getch();

        essential::gotoXY(56, 17);
        std::cout << a;

        //if the user press enter the program will consider it as skipped questioned
        if(int(a)==13)
        {
            essential::color(essential::BLUE);
            essential::gotoXY(55, 17);
            SetConsoleTextAttribute(h, 8);
            std::cout << "You skipped this Question";
        }

        else
        {
            if( a == questionList[index].getAnswer())
            {
                essential::gotoXY(55, 17);
                SetConsoleTextAttribute(h, 2);
                std::cout << "Congratulation You selected right option";
                userScore += 10; //incrementing 10 points each correct answers
            }

            else
            {
                essential::gotoXY(55, 17);
                SetConsoleTextAttribute(h, 4);
                std::cout << "You selected wrong option.";
                wrongAnswers++; //geting the wrong answers
            }
        }

        _getch();
        index++;
    }
```

}while(remainingTime != 0 && index < gettingFile.askUserQues); //if the remainingtime == 0 and the index reach the numeber of question that the user want the program will stop

```
        if (remainingTime == 0)
        {
            system("cls");
            sg.writeStudentData(userScore);
            gameOverScreen(menuAccessed);
        }

        else if (index == gettingFile.askUserQues)
        {
            system("cls");
            sg.writeStudentData(userScore);
            resultScreen(menuAccessed);
        }
    }

    void getQuestion()
    {
        gettingFile.openFile(questionFile);

        gettingFile.fillVector(questionList, questionFile);
    }

    HANDLE h = GetStdHandle(STD_OUTPUT_HANDLE);
};

void TimeTrialMode::gameOverScreen(void (*func)(void))
{
    essential::gotoXY(35, 10);
    SetConsoleTextAttribute(h, 15);
    std::cout << "Student Name: " << gameHistory::g_studentName;
    essential::gotoXY(70, 10);
    SetConsoleTextAttribute(h, 15);
    std::cout << "Student No. " << gameHistory::g_studentNumber;
    essential::gotoXY(35, 13);
    SetConsoleTextAttribute(h, 15);
    std::cout << "Your score ---> " << userScore;
    essential::gotoXY(70, 13);
    SetConsoleTextAttribute(h, 15);
    std::cout << sg.getFeedback(userScore);

    //call game over ascii art and also the border
    essential::asciiArtGameOver(func);
}

void TimeTrialMode::resultScreen(void (*func)(void))
{
```

```
        essential::gotoXY(35, 10);
        SetConsoleTextAttribute(h, 15);
        std::cout << "Student Name: " << gameHistory::g_studentName;
        essential::gotoXY(70, 10);
        SetConsoleTextAttribute(h, 15);
        std::cout << "Student No. " << gameHistory::g_studentNumber;
        essential::gotoXY(35, 13);
        SetConsoleTextAttribute(h, 15);
        std::cout << "Your score ---> " << userScore;
        essential::gotoXY(70, 12);
        SetConsoleTextAttribute(h, 15);
        std::cout << sg.getFeedback(userScore);
        essential::gotoXY(70, 14);
        SetConsoleTextAttribute(h, 15);
        std::cout << wrongAnswers;

        //call result ascii art and also the border
        essential::asciiArtResult(func);
    }
}

#endif
```

VIII. CONCLUSION

Understanding and applying advanced programming concepts in C++ empowered our group to develop a robust system. Despite the challenges, we successfully integrated these concepts into our program, ensuring compatibility and efficiency. Our Quiz Wizard system is accessible directly on the device, eliminating the need for online subscriptions. Students and teachers can freely assess knowledge without any additional costs. By leveraging advanced C++ programming, we optimized performance, enhanced data handling, and revolutionized offline learning experiences.