Code:

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
using namespace std;

const int MAX_PLAYERS = 100;

struct Player {
        string name;
        int score;
        int computerScore;
    // Default constructor
        Player()
          : name(""), score(0), computerScore(0) {}
   // Parameter constructor
         Player(const string& n, int s, int cS)
           : name(n), score(s), computerScore(cS) {}
};

void saveScores (const Player players[], int numPlayers) {
   ofstream file("scores.txt");
   if (file.is_open()) {
      for (int i = 0; i < numPlayers; ++i) {
```

```
            file << players[i].name << " "
                                    << players[i].score << " "
                 << players[i].computerScore << endl;
        }
        file.close();
        cout << "Scores saved successfully.\n";
    } else {
        cerr << "Unable to open file for saving scores.\n";
    }
}

void loadScores(Player players[], int& numPlayers) {
    ifstream file("scores.txt");
    if (file.is_open()) {
        numPlayers = 0;
        string name;
        int  score, computerScore;
        while (file >> name >> score >> computerScore &&
               numPlayers < MAX_PLAYERS) {
            players[numPlayers] =
                Player(name, score, computerScore);
            ++numPlayers;
        }
        file.close();
        cout << "Scores loaded successfully.\n";
    } else {
        cerr << "Unable to open file for loading scores.\n";
```

```
    }
}
void RSP(int choice) {
    switch (choice) {
    case 1:
        cout << "Rock.\n\n";
        cout << "    _______\n";
        cout << "--'        ____)\n";
        cout << "           (_____)   \n";
        cout << "           (_____)  \n";
        cout << "          (____) \n";
        cout << "--.__(___)\n\n";
        break;
    case 2:
        cout << "Paper.\n\n";
        cout << "    _______\n";
        cout << "---'         ____)____\n";
        cout << "                    _____)\n";
        cout << "               _______)\n";
        cout << "               _______)\n";
        cout << "---._____________)\n\n";
        break;

    case 3:
        cout << "Scissors.\n\n";
        cout << "    _______\n";
        cout << "---'      ____)____\n";
```

```
        cout << "              _______)\n";
        cout << "           ___________)\n";
        cout << "         (____)\n";
        cout << "---.__(___)\n\n";
        break;
    }
}

int main() {
    srand(time(0)); // Seed for random number generation
    Player players[MAX_PLAYERS];
    int numPlayers = 0;
    int computerScore = 0;
    int score = 0;
    loadScores(players, numPlayers);
    cout << "ROCK PAPER SCISSORS\n\n";
    while (true) {
        cout << "1. Play Game\n";
        cout << "2. View Scores\n";
        cout << "3. Quit\n";
        cout << "Enter your choice: ";
        int choice;
        cin >> choice;
        switch (choice) {
        case 1: {
            string playerName;
            cout << "Enter your name: ";
```

```
 cin >> playerName;
 while (true) {
    int userChoice;
    cout << playerName << ", let's play Rock-Paper-Scissors!\n\n";
    cout << "Enter your choice:\n";
    cout << "1 - Rock\n";
    cout << "2 - Paper\n";
    cout << "3 - Scissors\n\n";
    cin >> userChoice;
    int computerChoice = rand() % 3 + 1;
    cout << "You chose \n\n";
    RSP(userChoice);
    cout << "The computer chose \n\n";
    RSP(computerChoice);
    if ((userChoice == 1 && computerChoice == 3) ||
       (userChoice == 2 && computerChoice == 1) ||
       (userChoice == 3 && computerChoice == 2)) {
       cout << "You win!\n";
score++;
       bool playerFound = false;
       for (int i = 0; i < numPlayers; ++i) {
          if (players[i].name == playerName) {
             players[i].score=score;
             playerFound = true;
             break;
          }
       }
```

```
    if (!playerFound) {
        // If the player is not found, add a new player
        if (numPlayers < MAX_PLAYERS) {
            players[numPlayers] =
            Player(playerName, score, computerScore);
            ++numPlayers;
        } else {
            cerr << "Maximum number of players reached.\n";
        }
    }
} else if (userChoice == computerChoice) {
    cout << "It's a tie!\n";
} else {
    cout << "Computer wins!\n";
    computerScore++;
    bool playerFound = false;
    for (int i = 0; i < numPlayers; ++i) {
        if (players[i].name == playerName) {
            players[i].computerScore=computerScore;
            playerFound = true;
            break;
        }
    }
    if (!playerFound) {
        // If the player is not found, add a new player
        if (numPlayers < MAX_PLAYERS) {
            players[numPlayers] =
```

```
                    Player(playerName, score, computerScore);
                ++numPlayers;
            } else {
                cerr << "Maximum number of players reached.\n";
            }
        }
    }
    cout << "Do you want to play again? (1 for yes, 0 for no): ";
    int playAgain;
    cin >> playAgain;
    if (playAgain != 1) {
        computerScore = 0;
                    score = 0;

        break; // Exit the game loop
    }
  }
system("cls");
    break;
  }
  case 2: {
    // View Scores
    cout << endl;
    cout << "Scores are: \n";
    for (int i = 0; i < numPlayers; ++i) {
      cout << players[i].name << ": " << players[i].score
          << ", Computer: " << players[i].computerScore << ")\n";
```

```
            }
            break;
        }
        case 3: {
            // Save Scores and Quit
            saveScores(players, numPlayers);
            cout << "Thanks for playing!\n";
            return 0;
        }
        default:
            cout << "Invalid choice. Please enter a valid option.\n";
        }
    }
    return 0;
}
```

Explanation:

#include <iostream>: Input/output library for basic console input and output operations.

#include <fstream>: File stream library for reading from and writing to files.

#include <cstdlib>: C standard library for general-purpose functions, including random number generation.

#include <ctime>: C time library for time-related functions.

using namespace std;: Allows usage of elements from the std namespace (standard C++ library) without explicit namespace specification.

const int MAX_PLAYERS = 100; : limited the amount of total players to 100

struct Player :

This is a C++ struct definition named Player, which defines a blueprint for an object representing a player in a game or a similar scenario.

It contains three member variables:

string name;: Represents the name of the player, using the string data type from the C++ standard library.

int score;: Represents the player's score in the game.

int computerScore;: Represents the computer's score in the game.

Default constructor (Player()):

- Initializes the name to an empty string " ".
- Initializes the score and computerScore to zero using the member initialization list (designated by : name(""), score(0), computerScore(0) {}). This constructor initializes all member variables to their default values.

Parameterized constructor (Player(const string& n, int s, int cS)):

- Accepts three parameters: a string for the player's name (n), an int for the player's score (s), and an int for the computer's score (cS).
- Initializes the name to the passed n.
- Initializes the score to the passed s.
- Initializes the computerScore to the passed cS. This constructor allows initialization of the Player object with specific values for each member variable.

void loadScores;

This C++ function, saveScores, takes an array of Player objects (players[]) and the number of players (numPlayers) as parameters. Its purpose is to save the scores of players to a file named "scores.txt".

Here's a breakdown of what this function does:

ofstream file("scores.txt");: Creates an output file stream object named file and opens a file named "scores.txt" for writing.

if (file.is_open()) { ... }: Checks if the file was successfully opened. If the file is open:

- The function enters a loop that iterates through each player in the **players[]** array.

- Inside the loop, the function writes the **player's name, score, and computerScore** to the file using the file stream. It uses the << operator to write each attribute followed by a space, and then adds a newline (endl) to move to the next line for the next player's data.
- After writing the data for all players, the file is closed using **file.close().**
- If the file was opened and data was successfully written, it displays a message **"Scores saved successfully." using cout.**
- If the file couldn't be opened, an error **message "Unable to open file for saving scores." is displayed using cerr.**
- This function essentially takes the player data provided in the array and writes it to a file named **"scores.txt"** in a structured manner, separating each player's information with a newline for easy retrieval and readability.

void RSP(int choice)

This C++ function RSP(int choice) represents a simplified version of the game "Rock, Paper, Scissors". It takes an integer choice as input and displays ASCII art representing the corresponding choice made by the player or the computer in the game.

The function uses a switch statement based on the value of choice:

- If choice is 1, it displays an ASCII art representation of "Rock".
- If choice is 2, it displays an ASCII art representation of "Paper".
- If choice is 3, it displays an ASCII art representation of "Scissors".

Each case in the switch statement prints out the corresponding choice's ASCII art using multiple cout statements. The ASCII art represents the visual appearance of the chosen item (rock, paper, or scissors) using text characters in a stylized format.

int main() :

This C++ **main()** function seems to implement a Rock-Paper-Scissors game with additional functionalities to manage player scores and choices. Here's a breakdown of what the code does:

Initialization:

srand(time(0));: Seeds the random number generator using the current time to generate random numbers later in the game.

Defines arrays and variables to manage player data, including **Player players[MAX_PLAYERS], numPlayers, computerScore, and score.**

Game Loop:

Displays a menu to the user with options to play the game, view scores, or quit the game.

Based on the user's choice, it executes different actions:

- **Play Game (case 1):**

    1. Takes player's name input and starts the Rock-Paper-Scissors game.
    2. Repeats the game until the player decides to stop.
    3. Tracks the player's and computer's choices, determines the winner, and updates scores accordingly.
    4. Manages player data by updating existing player scores or adding new players to the players array.

- **View Scores (case 2):**

    1. Displays the current scores of players and the computer.
    2. Save Scores and Quit (case 3):
    3. Saves the scores to a file using saveScores() function and exits the program.

Game Logic:

The game follows standard Rock-Paper-Scissors rules, compares the user's choice with the computer's choice, and determines the winner or a tie.

Updates the scores and manages player data based on the game outcome.

Provides an option to the player to continue playing or quit after each round.

Player Data Management:

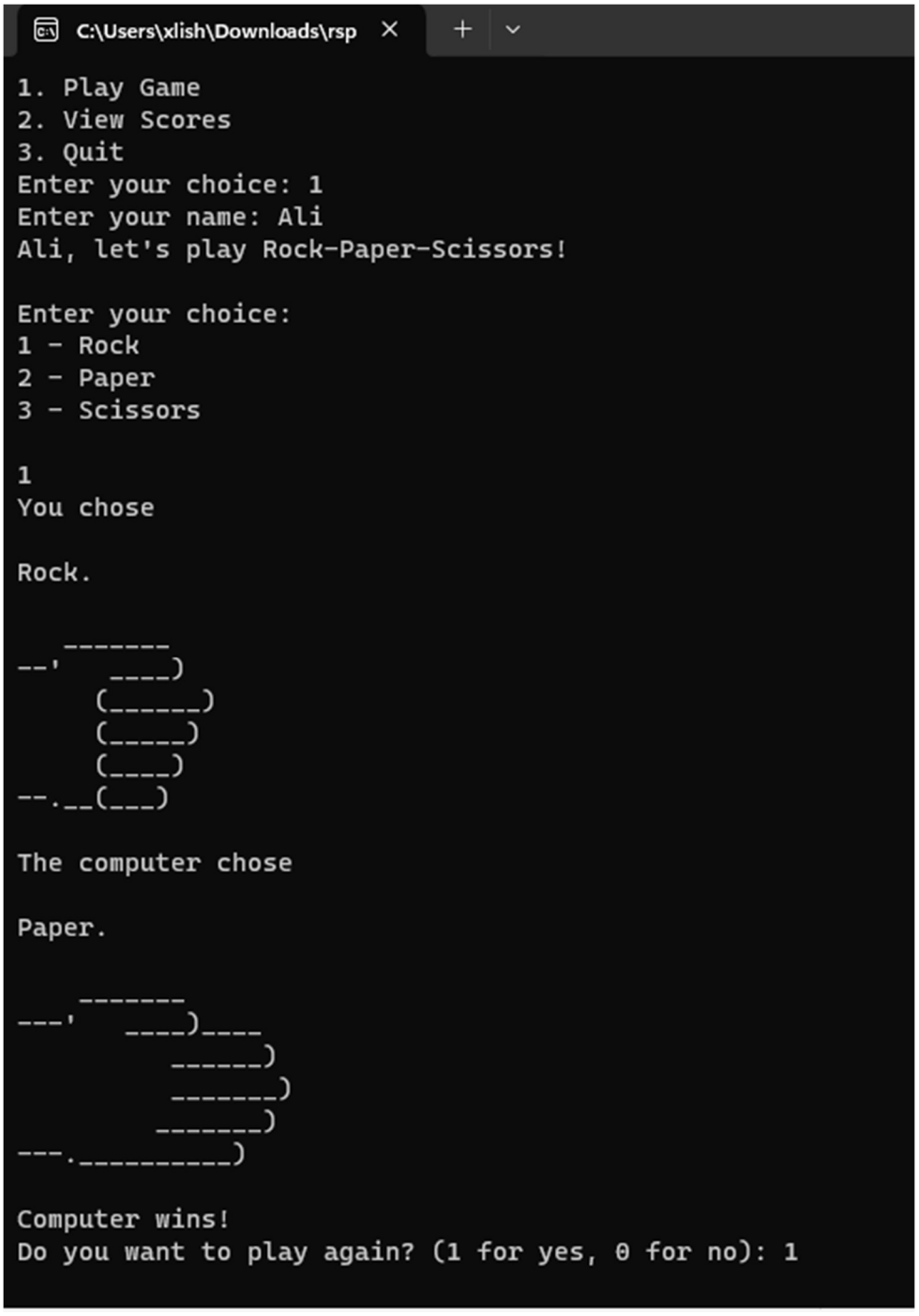
Manages player data by checking existing player records and updating their scores. If the player is new, it adds the player's data to the players array.

Limits the number of players to MAX_PLAYERS to prevent exceeding the array size.

Input Validation:

Includes basic input validation for user choices.

Output:



Note: The above operation was done multiple times to get the below score results

```
1. Play Game
2. View Scores
3. Quit
Enter your choice: 2

Scores are:
Ali: 3, Computer: 3
ahmed: 5, Computer: 2
tariq: 1, Computer: 2
Ahmed: 7, Computer: 8
Tariq: 11, Computer: 14

1. Play Game
2. View Scores
3. Quit
Enter your choice: 3
Scores saved successfully.
Thanks for playing!

--------------------------------
Process exited after 99.41 seconds with return value 0
Press any key to continue . . .
```