Overview:

This project deals with the chess game. This program includes many features like register player, login player, show registered players in the game, play game and exit the game. In play game, there is also an option to play with the computer or to play with the opponent player. In registered player option, the game asks for the username and password. In login player option of game, the player enter its username and its password that it uses during registration and if he enters wrong password or username, the game displays error. When the second user tries to register himself with the password that the other players have used, then he will have to change his password to register and to login in the game. When we press want to exit either during the game or at the end, there is also an option to exit the game.

Chess Game

The features that we have used in this program are as follows:

Header files:

- #include <iostream>
- #include <stack>
- #include <queue>
- #include <vector>
- #include <string>
- #include <cstdlib> // for random computer move
- #include <ctime>   // for random seed

Structures:

- struct MoveNode
- struct PlayerNode
- struct PieceNode

Classes and Functions used in them:

class Player

- Player()

class ChessBoard

- ChessBoard()
- displayBoard()
- movePiece()
- getPositionFromInput()
- computerMove()

Class MoveHistory

- addMove()
- displayMoves()

class PlayerTurnQueue

- PlayerTurnQueue()
- addPlayer()
- nextTurn()

class PieceHistory

- PieceHistory()
- addPiece()
- undoMove()
- redoMove()

class GameStateStack

- saveState()
- undoState()

class MoveQueue

- addMove()
- executeMove()

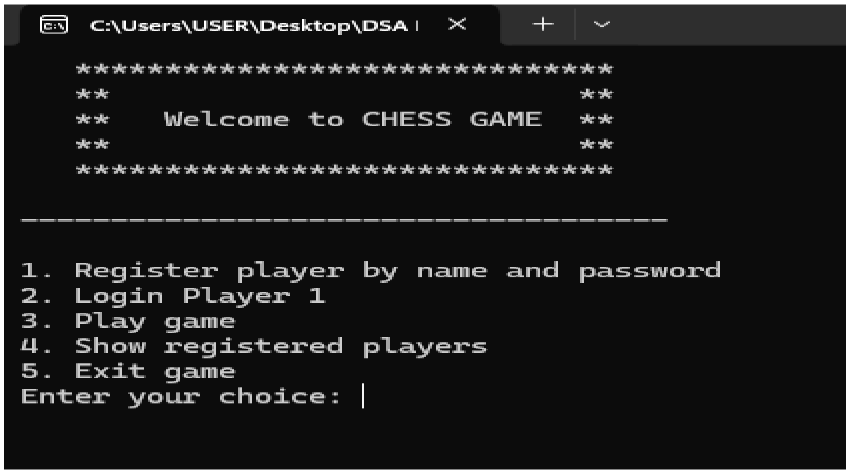
Other Global Functions:

displayMainMenu()

- registerPlayer()
- loginPlayer()
- computerMove()
- isLastRowEmpty()
- playGame()

Main Function:

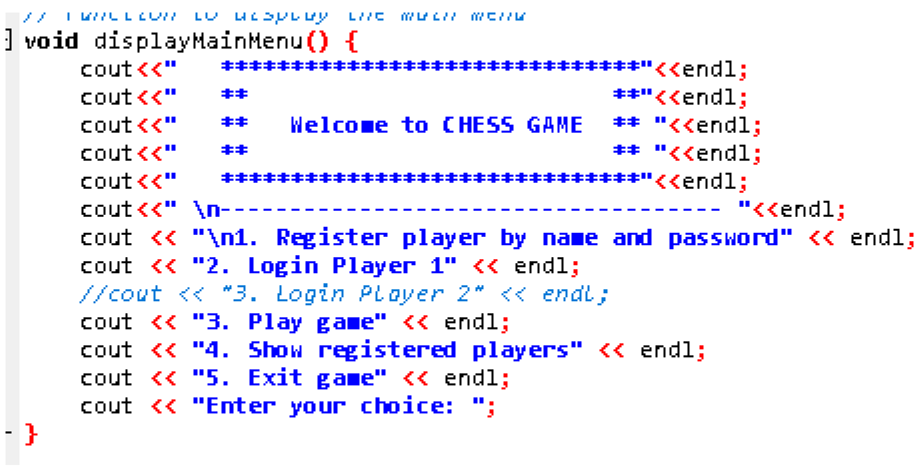
In the main function we use while loop in which user enter its choice according to game display menu. If the choice is 1, the game will ask for registration.if the choice is 2, the game will ask for login. If the choice is 3, the game will ask to play. If the choice is 4, the game will displayed the registered players in the game. If the choice is 5 the loop will end and the game exits.



```
C:\Users\USER\Desktop\DSA

   ******************************
   **                          **
   **   Welcome to CHESS GAME  **
   **                          **
   ******************************

------------------------------------

1. Register player by name and password
2. Login Player 1
3. Play game
4. Show registered players
5. Exit game
Enter your choice: |
```

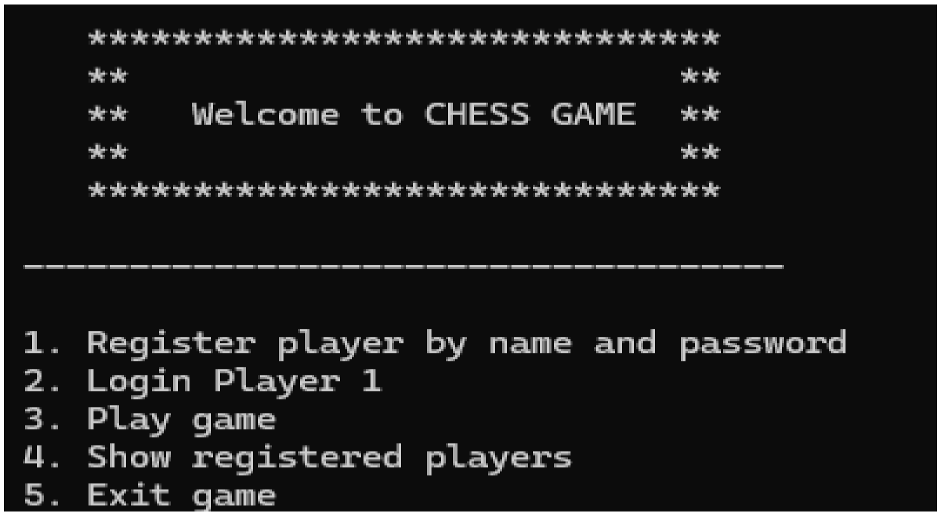
Scenario:

- **When we enter the game, the menu will be displayed. It asks for user choice.**



```
// function to display the main menu
void displayMainMenu() {
    cout<<"   ******************************"<<endl;
    cout<<"   **                          **"<<endl;
    cout<<"   **   Welcome to CHESS GAME  ** "<<endl;
    cout<<"   **                          ** "<<endl;
    cout<<"   ******************************"<<endl;
    cout<<" \n-----------------------------------  "<<endl;
    cout << "\n1. Register player by name and password" << endl;
    cout << "2. Login Player 1" << endl;
    //cout << "3. Login Player 2" << endl;
    cout << "3. Play game" << endl;
    cout << "4. Show registered players" << endl;
    cout << "5. Exit game" << endl;
    cout << "Enter your choice: ";
}
```

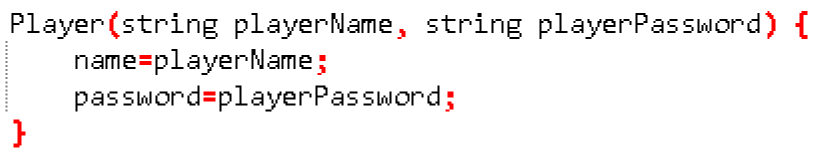
Output:



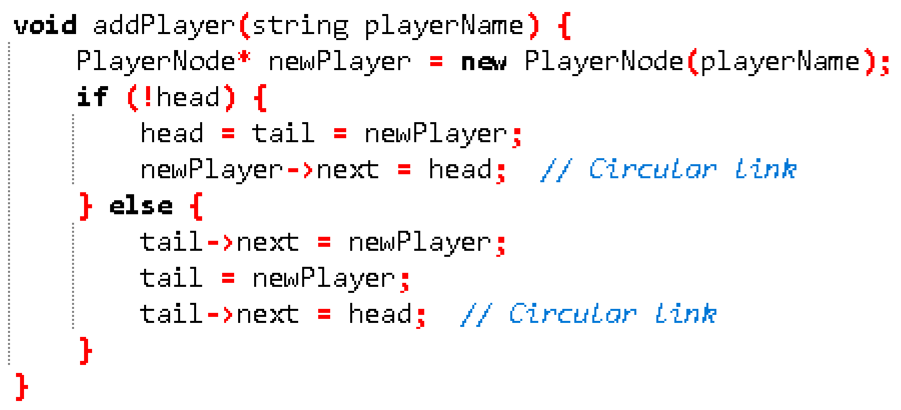
```
   ******************************
   **                          **
   **   Welcome to CHESS GAME  **
   **                          **
   ******************************

-----------------------------------

1. Register player by name and password
2. Login Player 1
3. Play game
4. Show registered players
5. Exit game
```

❖ If the choice is one, the game will ask for registration;

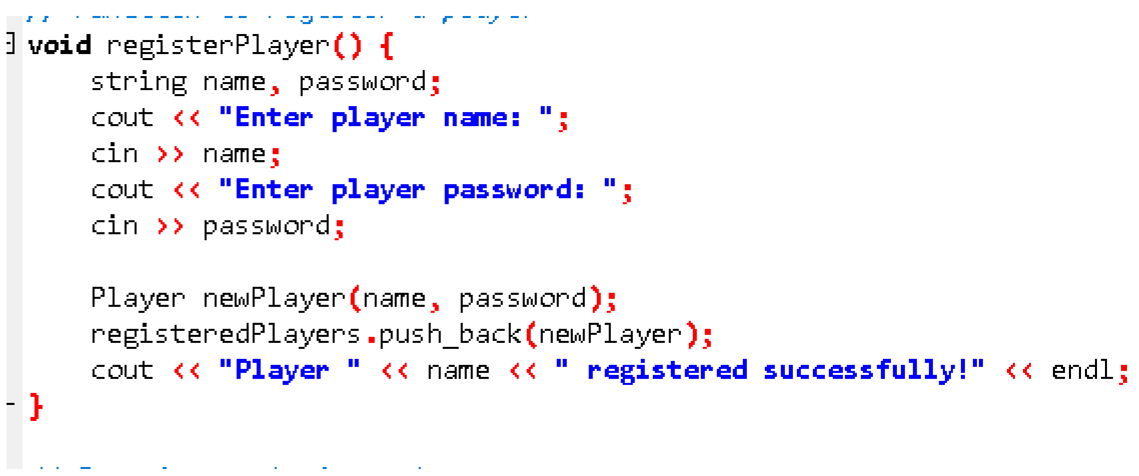
I have made functions in different classes, the functions required for registration are as follows;



```
Player(string playerName, string playerPassword) {
    name=playerName;
    password=playerPassword;
}
```

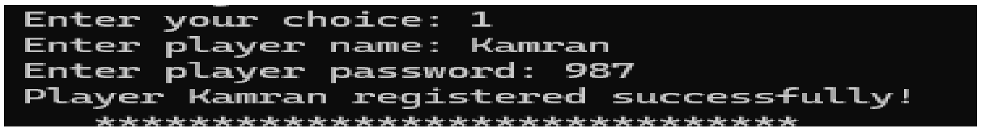


```cpp
void addPlayer(string playerName) {
    PlayerNode* newPlayer = new PlayerNode(playerName);
    if (!head) {
        head = tail = newPlayer;
        newPlayer->next = head;  // Circular link
    } else {
        tail->next = newPlayer;
        tail = newPlayer;
        tail->next = head;  // Circular link
    }
}
```



```cpp
void registerPlayer() {
    string name, password;
    cout << "Enter player name: ";
    cin >> name;
    cout << "Enter player password: ";
    cin >> password;

    Player newPlayer(name, password);
    registeredPlayers.push_back(newPlayer);
    cout << "Player " << name << " registered successfully!" << endl;
}
```

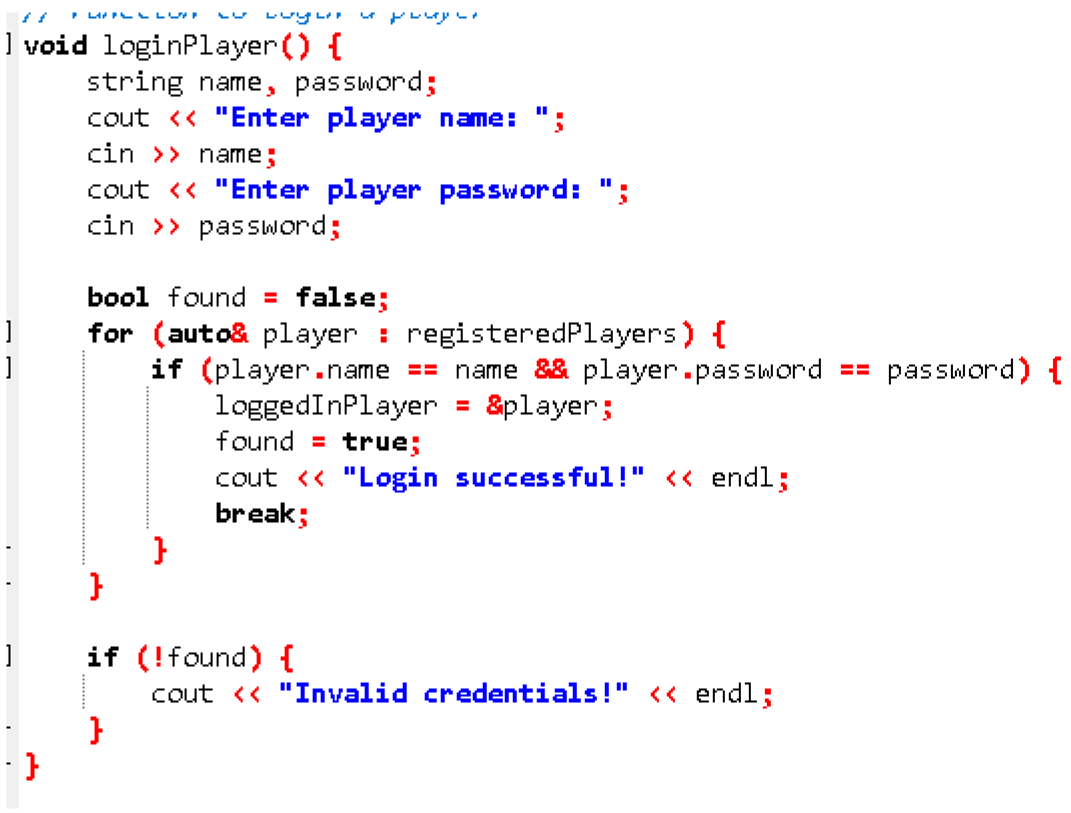
Result:



```
Enter your choice: 1
Enter player name: Kamran
Enter player password: 987
Player Kamran registered successfully!
```

❖ When we press 2, it will ask for login player.

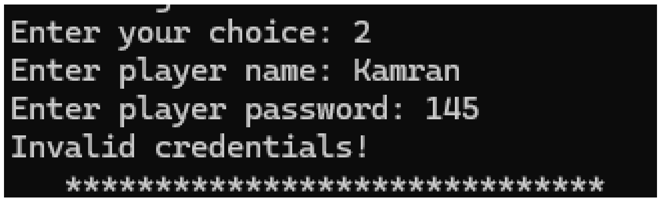
The functions needed for login players are as follows;
In this function, I have started the registered player function in the for loop, it means that for loop will con tine until registered player function will not end.



```
// Function to login a player
void loginPlayer() {
    string name, password;
    cout << "Enter player name: ";
    cin >> name;
    cout << "Enter player password: ";
    cin >> password;

    bool found = false;
    for (auto& player : registeredPlayers) {
        if (player.name == name && player.password == password) {
            loggedInPlayer = &player;
            found = true;
            cout << "Login successful!" << endl;
            break;
        }
    }

    if (!found) {
        cout << "Invalid credentials!" << endl;
    }
}
```

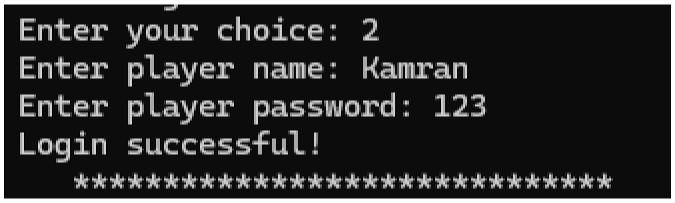
Output:

- When password does not match



```
Enter your choice: 2
Enter player name: Kamran
Enter player password: 145
Invalid credentials!
   *****************************
```

- When password matches;



```
Enter your choice: 2
Enter player name: Kamran
Enter player password: 123
Login successful!
   *****************************
```

❖ When we press 3, it shows two options;

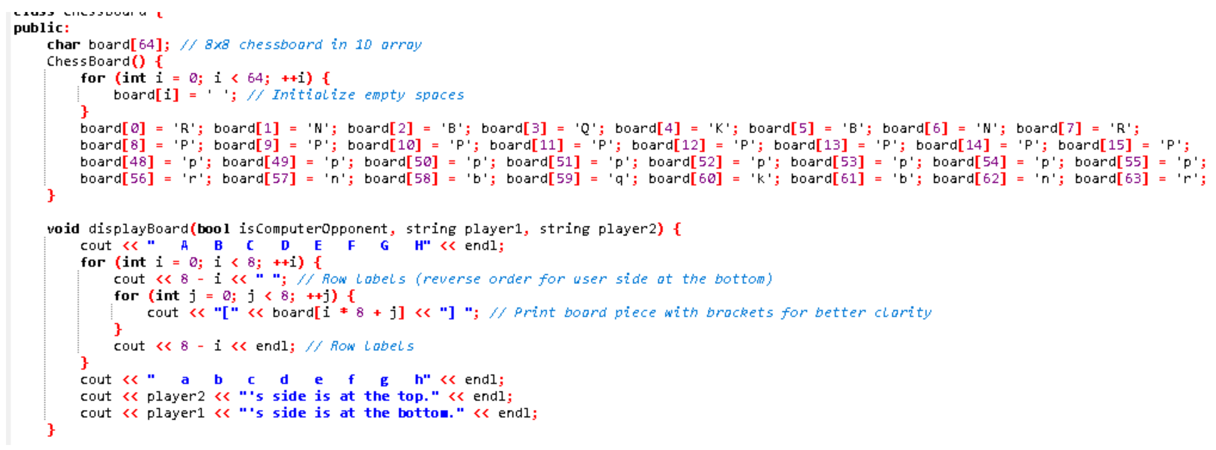
1. Play with computer
2. Play with opponent player

A function will be called;

```
void playGame() {
    string playerMove;

    ChessBoard board;
    MoveHistory moveHistory;
    PlayerTurnQueue turnQueue;
    PieceHistory pieceHistory;
    GameStateStack gameStateStack;
    MoveQueue moveQueue;

    string playerName1, playerName2;
    cout << "Enter Player 1 name: ";
    cin >> playerName1;
    cout << "Enter Player 2 name (uppercase moves): ";
    cin >> playerName2;

    int gameChoice;
    cout << "1. Play with computer" << endl;
    cout << "2. Play with another player" << endl;
    cout << "Enter your choice: ";
    cin >> gameChoice;
```

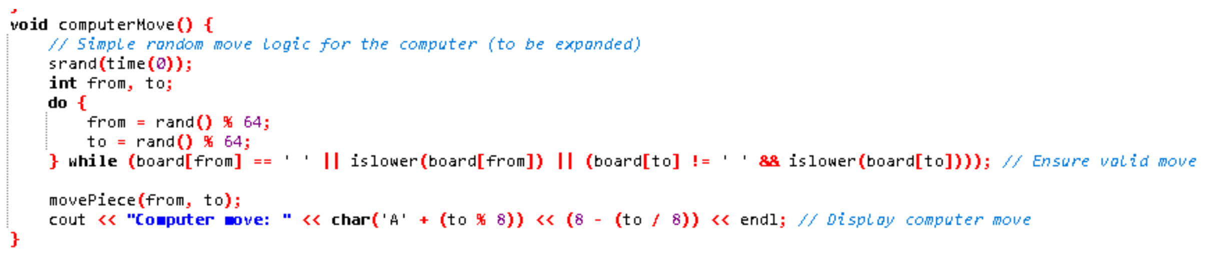
1.Play with computer

When the game is started, it shows the chessboard.



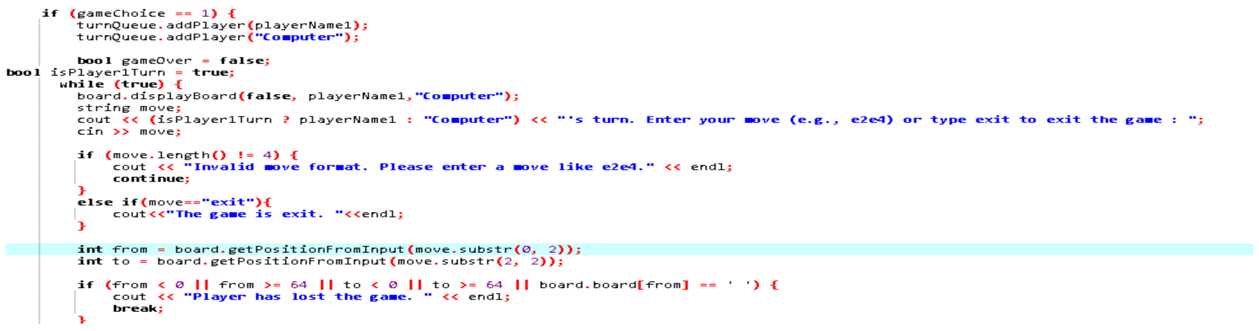
```
class ChessBoard {
public:
    char board[64]; // 8x8 chessboard in 1D array
    ChessBoard() {
        for (int i = 0; i < 64; ++i) {
            board[i] = ' '; // Initialize empty spaces
        }
        board[0] = 'R'; board[1] = 'N'; board[2] = 'B'; board[3] = 'Q'; board[4] = 'K'; board[5] = 'B'; board[6] = 'N'; board[7] = 'R';
        board[8] = 'P'; board[9] = 'P'; board[10] = 'P'; board[11] = 'P'; board[12] = 'P'; board[13] = 'P'; board[14] = 'P'; board[15] = 'P';
        board[48] = 'p'; board[49] = 'p'; board[50] = 'p'; board[51] = 'p'; board[52] = 'p'; board[53] = 'p'; board[54] = 'p'; board[55] = 'p';
        board[56] = 'r'; board[57] = 'n'; board[58] = 'b'; board[59] = 'q'; board[60] = 'k'; board[61] = 'b'; board[62] = 'n'; board[63] = 'r';
    }

    void displayBoard(bool isComputerOpponent, string player1, string player2) {
        cout << "    A   B   C   D   E   F   G   H" << endl;
        for (int i = 0; i < 8; ++i) {
            cout << 8 - i << " "; // Row labels (reverse order for user side at the bottom)
            for (int j = 0; j < 8; ++j) {
                cout << "[" << board[i * 8 + j] << "] "; // Print board piece with brackets for better clarity
            }
            cout << 8 - i << endl; // Row labels
        }
        cout << "    a   b   c   d   e   f   g   h" << endl;
        cout << player2 << "'s side is at the top." << endl;
        cout << player1 << "'s side is at the bottom." << endl;
    }
```

- When we play with computer, the player do his move and after that the computer use his elements from the upper side.
  Computer move function has been built in this code and I use cstdlib header files for random computer move.

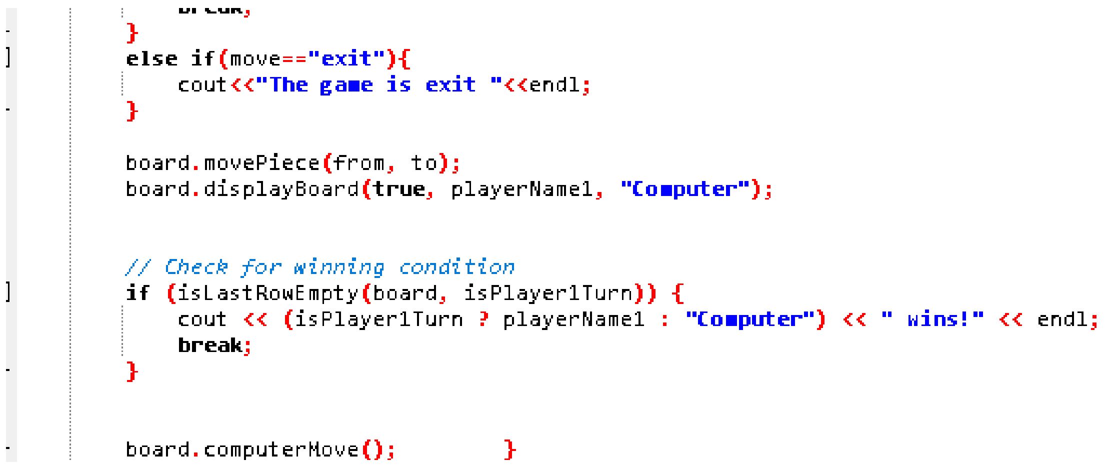


```
void computerMove() {
    // Simple random move logic for the computer (to be expanded)
    srand(time(0));
    int from, to;
    do {
        from = rand() % 64;
        to = rand() % 64;
    } while (board[from] == ' ' || islower(board[from]) || (board[to] != ' ' && islower(board[to]))); // Ensure valid move

    movePiece(from, to);
    cout << "Computer move: " << char('A' + (to % 8)) << (8 - (to / 8)) << endl; // Display computer move
}
```

- The play with computer function will call when I play game with the computer.

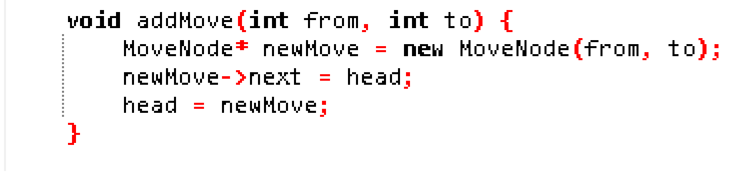


```
    if (gameChoice == 1) {
        turnQueue.addPlayer(playerName1);
        turnQueue.addPlayer("Computer");

        bool gameOver = false;
bool isPlayer1Turn = true;
     while (true) {
        board.displayBoard(false, playerName1,"Computer");
        string move;
        cout << (isPlayer1Turn ? playerName1 : "Computer") << "'s turn. Enter your move (e.g., e2e4) or type exit to exit the game : ";
        cin >> move;

        if (move.length() != 4) {
            cout << "Invalid move format. Please enter a move like e2e4." << endl;
            continue;
        }
        else if(move=="exit"){
            cout<<"The game is exit. "<<endl;
        }

        int from = board.getPositionFromInput(move.substr(0, 2));
        int to = board.getPositionFromInput(move.substr(2, 2));

        if (from < 0 || from >= 64 || to < 0 || to >= 64 || board.board[from] == ' ') {
            cout << "Player has lost the game. " << endl;
            break;
        }
```



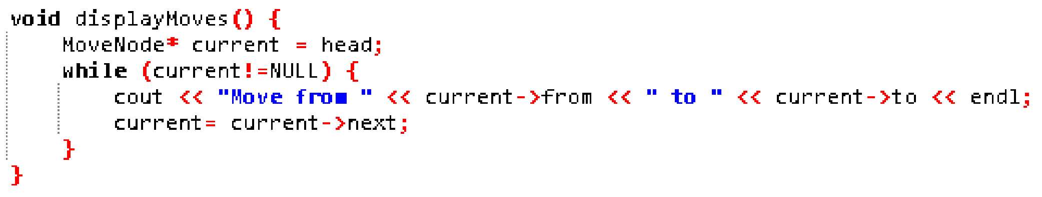
```
    }
    else if(move=="exit"){
        cout<<"The game is exit "<<endl;
    }

    board.movePiece(from, to);
    board.displayBoard(true, playerName1, "Computer");


    // Check for winning condition
    if (isLastRowEmpty(board, isPlayer1Turn)) {
        cout << (isPlayer1Turn ? playerName1 : "Computer") << " wins!" << endl;
        break;
    }


    board.computerMove();        }
```

- There is also a function that first add the move of player.



```
void addMove(int from, int to) {
    MoveNode* newMove = new MoveNode(from, to);
    newMove->next = head;
    head = newMove;
}
```

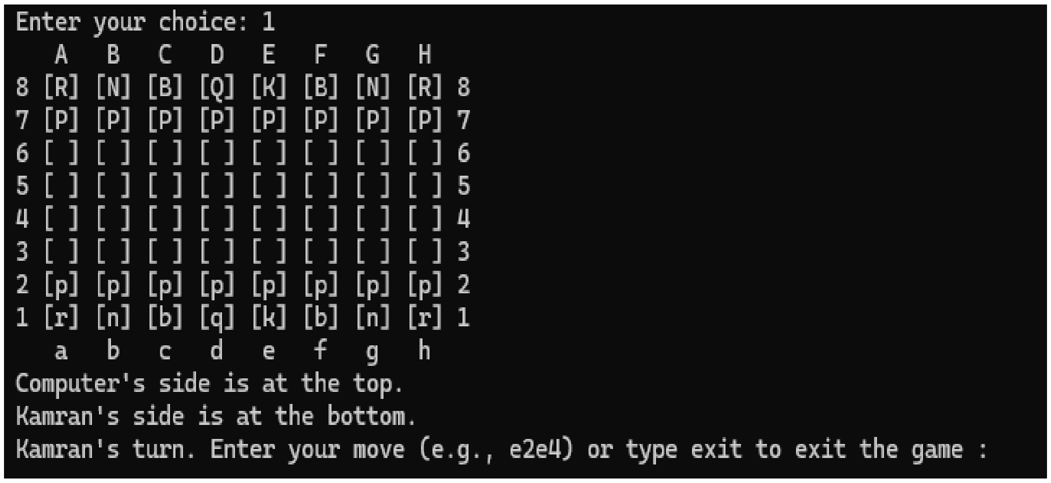
- There is a function to display the moves of the players.



```
void displayMoves() {
    MoveNode* current = head;
    while (current!=NULL) {
        cout << "Move from " << current->from << " to " << current->to << endl;
        current= current->next;
    }
}
```

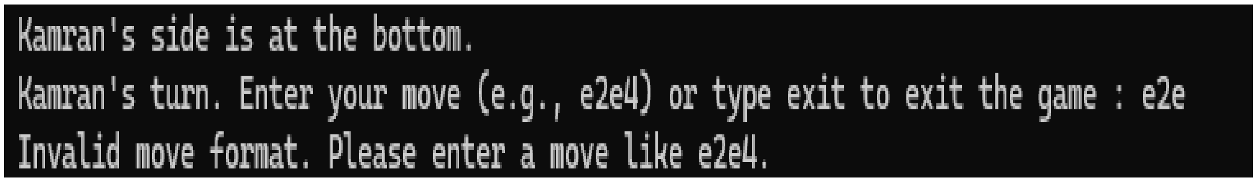
Output:

- When I press 1, it starts the game with the computer.



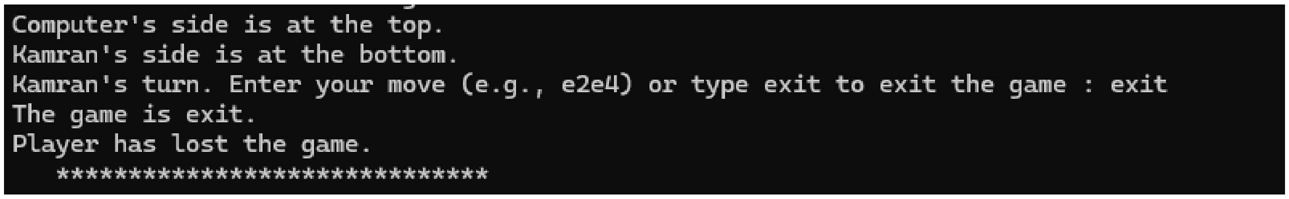
```
Enter your choice: 1
   A   B   C   D   E   F   G   H
8 [R] [N] [B] [Q] [K] [B] [N] [R] 8
7 [P] [P] [P] [P] [P] [P] [P] [P] 7
6 [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] 6
5 [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] 5
4 [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] 4
3 [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] 3
2 [p] [p] [p] [p] [p] [p] [p] [p] 2
1 [r] [n] [b] [q] [k] [b] [n] [r] 1
   a   b   c   d   e   f   g   h
Computer's side is at the top.
Kamran's side is at the bottom.
Kamran's turn. Enter your move (e.g., e2e4) or type exit to exit the game :
```

- The game tell me to do valid move and if I do the wrong move, it will display the error;



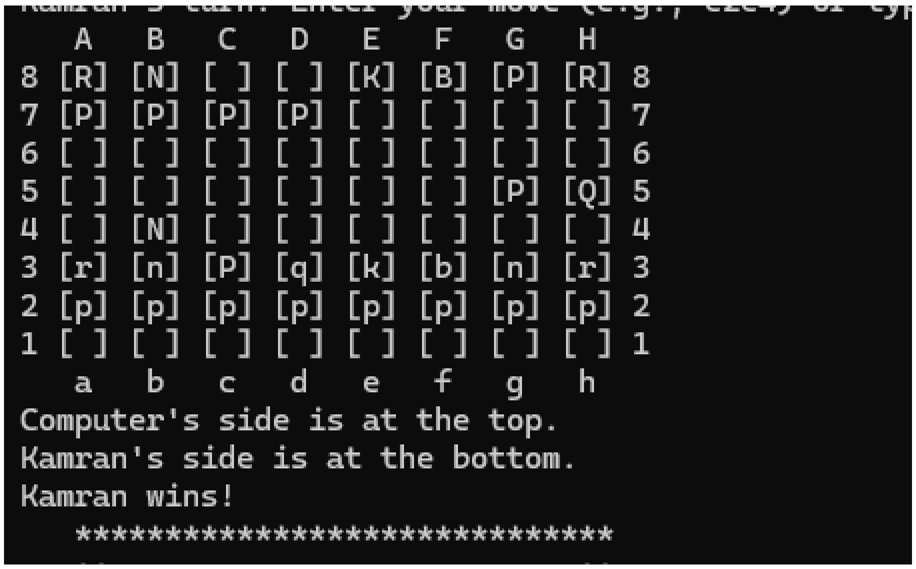
```
Kamran's side is at the bottom.
Kamran's turn. Enter your move (e.g., e2e4) or type exit to exit the game : e2e
Invalid move format. Please enter a move like e2e4.
```

- Whenever I want to quit the game during playing, I only type exit and the game will be exit y displaying the message that you have lost the game;



```
Computer's side is at the top.
Kamran's side is at the bottom.
Kamran's turn. Enter your move (e.g., e2e4) or type exit to exit the game : exit
The game is exit.
Player has lost the game.
   ********************************
```

- I have made the winning condition in this game that the player whose last row will be emptied first will won the game;

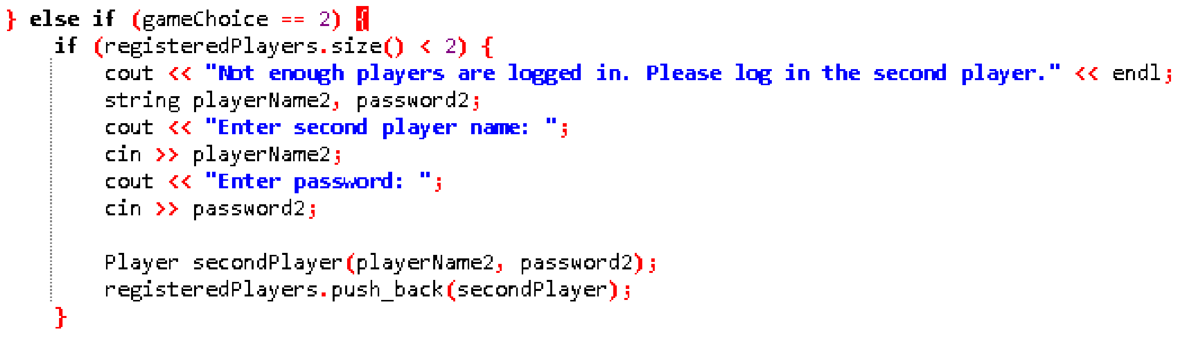


```
Kamran's turn. Enter your move (e.g., e2e4) or typ
   A   B   C   D   E   F   G   H
8 [R] [N] [ ] [ ] [K] [B] [P] [R] 8
7 [P] [P] [P] [P] [ ] [ ] [ ] [ ] 7
6 [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] 6
5 [ ] [ ] [ ] [ ] [ ] [ ] [P] [Q] 5
4 [ ] [N] [ ] [ ] [ ] [ ] [ ] [ ] 4
3 [r] [n] [P] [q] [k] [b] [n] [r] 3
2 [p] [p] [p] [p] [p] [p] [p] [p] 2
1 [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] 1
   a   b   c   d   e   f   g   h
Computer's side is at the top.
Kamran's side is at the bottom.
Kamran wins!
   ******************************
```

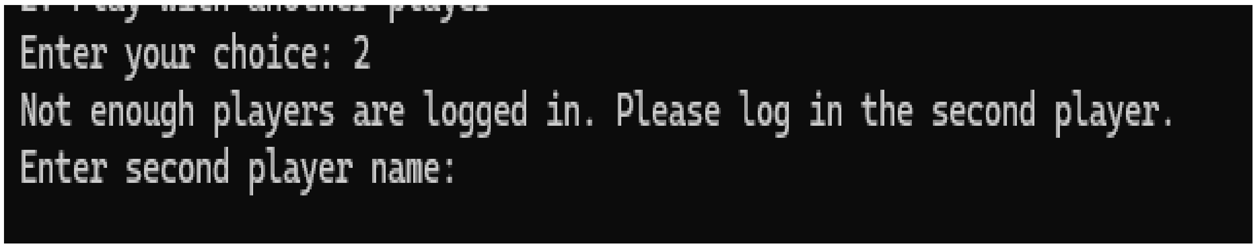
Here, you can see that I have won the game because I have emptied my last row first.

2. **Play with another player**

- When I press 2 to play with another player and if I have only login, it first ask for second player login.

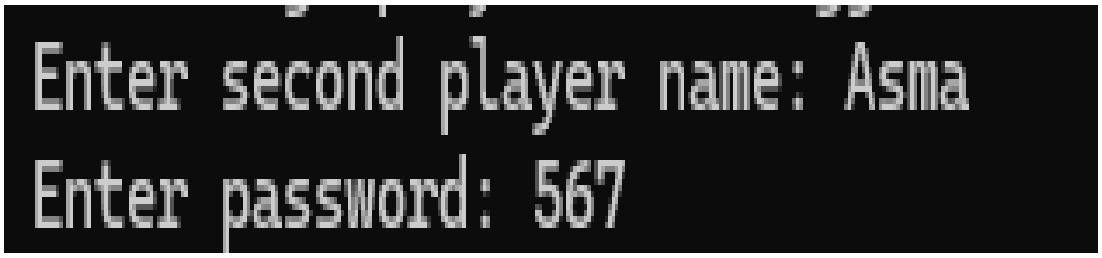


```
} else if (gameChoice == 2) {
    if (registeredPlayers.size() < 2) {
        cout << "Not enough players are logged in. Please log in the second player." << endl;
        string playerName2, password2;
        cout << "Enter second player name: ";
        cin >> playerName2;
        cout << "Enter password: ";
        cin >> password2;

        Player secondPlayer(playerName2, password2);
        registeredPlayers.push_back(secondPlayer);
    }
```



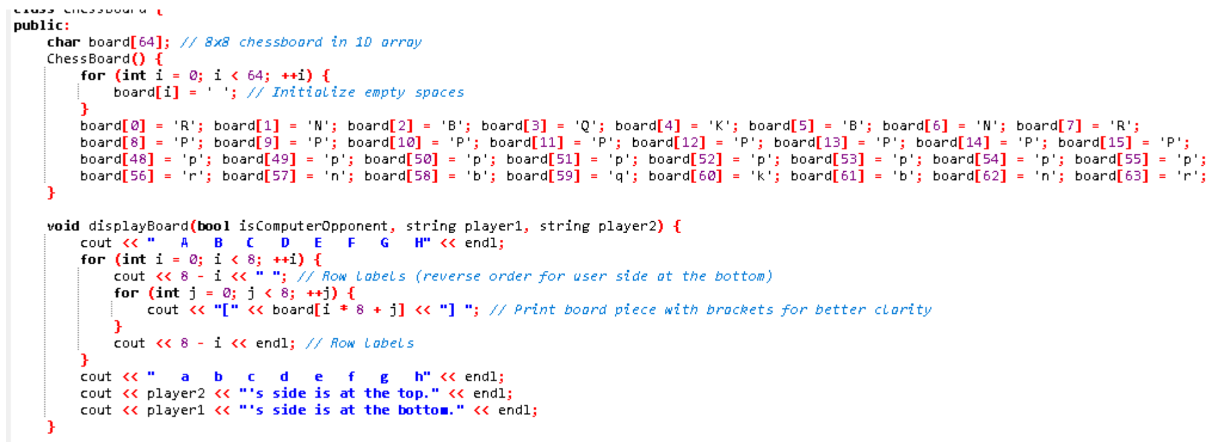
```
Enter your choice: 2
Not enough players are logged in. Please log in the second player.
Enter second player name:
```

- Now we first login the second player,



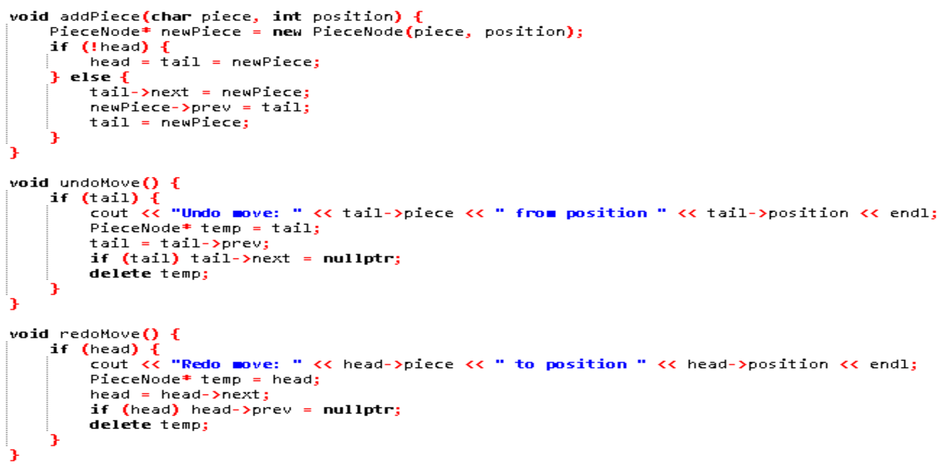
```
Enter second player name: Asma
Enter password: 567
```

- When the game is started, it shows the chessboard.



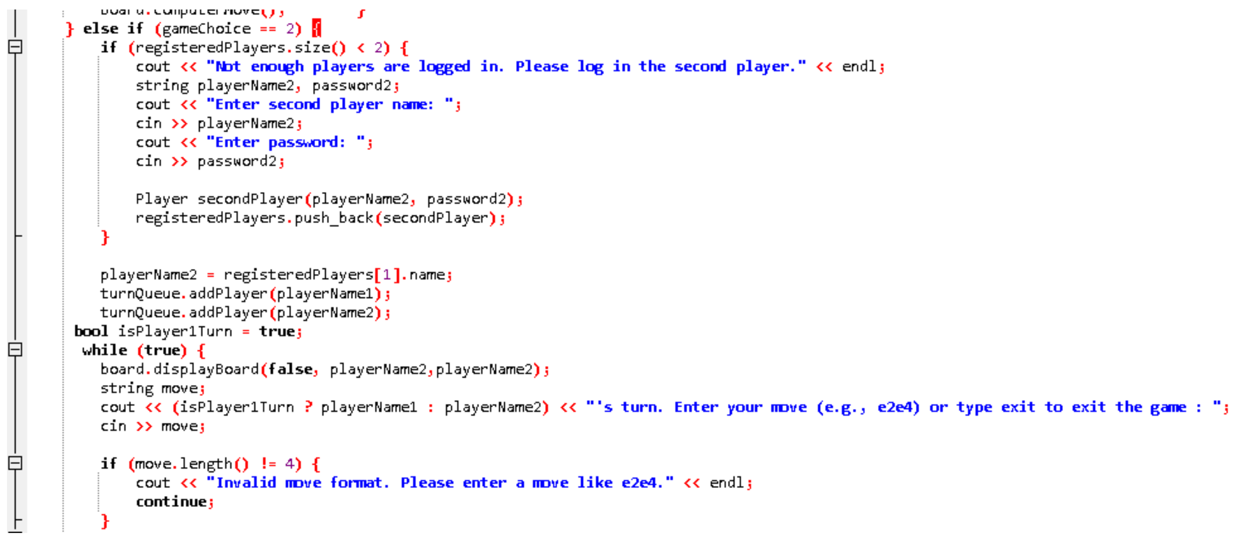
```cpp
public:
    char board[64]; // 8x8 chessboard in 1D array
    ChessBoard() {
        for (int i = 0; i < 64; ++i) {
            board[i] = ' '; // Initialize empty spaces
        }
        board[0] = 'R'; board[1] = 'N'; board[2] = 'B'; board[3] = 'Q'; board[4] = 'K'; board[5] = 'B'; board[6] = 'N'; board[7] = 'R';
        board[8] = 'P'; board[9] = 'P'; board[10] = 'P'; board[11] = 'P'; board[12] = 'P'; board[13] = 'P'; board[14] = 'P'; board[15] = 'P';
        board[48] = 'p'; board[49] = 'p'; board[50] = 'p'; board[51] = 'p'; board[52] = 'p'; board[53] = 'p'; board[54] = 'p'; board[55] = 'p';
        board[56] = 'r'; board[57] = 'n'; board[58] = 'b'; board[59] = 'q'; board[60] = 'k'; board[61] = 'b'; board[62] = 'n'; board[63] = 'r';
    }

    void displayBoard(bool isComputerOpponent, string player1, string player2) {
        cout << "   A   B   C   D   E   F   G   H" << endl;
        for (int i = 0; i < 8; ++i) {
            cout << 8 - i << " "; // Row labels (reverse order for user side at the bottom)
            for (int j = 0; j < 8; ++j) {
                cout << "[" << board[i * 8 + j] << "] "; // Print board piece with brackets for better clarity
            }
            cout << 8 - i << endl; // Row labels
        }
        cout << "   a   b   c   d   e   f   g   h" << endl;
        cout << player2 << "'s side is at the top." << endl;
        cout << player1 << "'s side is at the bottom." << endl;
    }
```

- When we play with player, the player do his move and after that the player use his elements from the upper side.
  player move function has been built in this code and I use vector and stack header files to access that function and use this function.

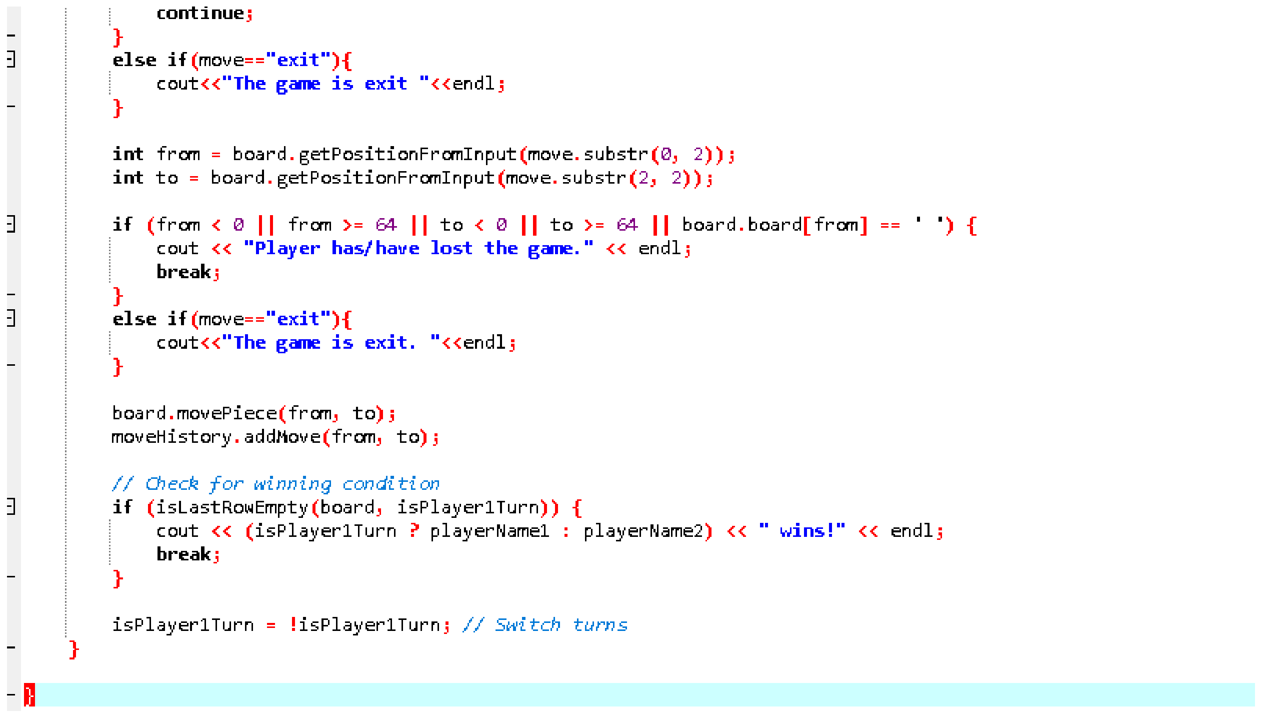


```cpp
void addPiece(char piece, int position) {
    PieceNode* newPiece = new PieceNode(piece, position);
    if (!head) {
        head = tail = newPiece;
    } else {
        tail->next = newPiece;
        newPiece->prev = tail;
        tail = newPiece;
    }
}

void undoMove() {
    if (tail) {
        cout << "Undo move: " << tail->piece << " from position " << tail->position << endl;
        PieceNode* temp = tail;
        tail = tail->prev;
        if (tail) tail->next = nullptr;
        delete temp;
    }
}

void redoMove() {
    if (head) {
        cout << "Redo move: " << head->piece << " to position " << head->position << endl;
        PieceNode* temp = head;
        head = head->next;
        if (head) head->prev = nullptr;
        delete temp;
    }
}
```

- The play with player function will call when I play game with the player.

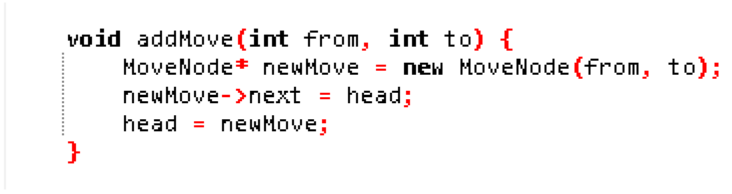


```
    board.computerMove();
} else if (gameChoice == 2) {
    if (registeredPlayers.size() < 2) {
        cout << "Not enough players are logged in. Please log in the second player." << endl;
        string playerName2, password2;
        cout << "Enter second player name: ";
        cin >> playerName2;
        cout << "Enter password: ";
        cin >> password2;

        Player secondPlayer(playerName2, password2);
        registeredPlayers.push_back(secondPlayer);
    }

    playerName2 = registeredPlayers[1].name;
    turnQueue.addPlayer(playerName1);
    turnQueue.addPlayer(playerName2);
 bool isPlayer1Turn = true;
  while (true) {
    board.displayBoard(false, playerName2,playerName2);
    string move;
    cout << (isPlayer1Turn ? playerName1 : playerName2) << "'s turn. Enter your move (e.g., e2e4) or type exit to exit the game : ";
    cin >> move;

    if (move.length() != 4) {
        cout << "Invalid move format. Please enter a move like e2e4." << endl;
        continue;
    }
```



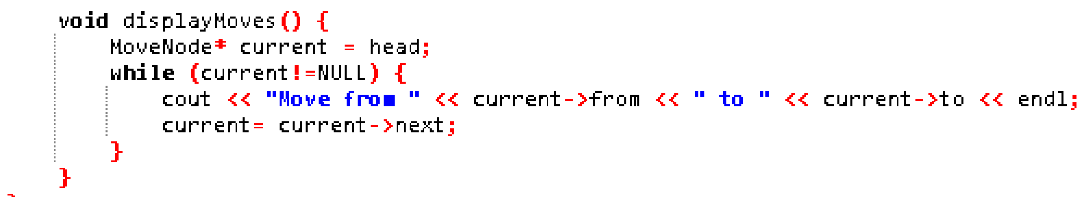
```
        continue;
    }
    else if(move=="exit"){
        cout<<"The game is exit "<<endl;
    }

    int from = board.getPositionFromInput(move.substr(0, 2));
    int to = board.getPositionFromInput(move.substr(2, 2));

    if (from < 0 || from >= 64 || to < 0 || to >= 64 || board.board[from] == ' ') {
        cout << "Player has/have lost the game." << endl;
        break;
    }
    else if(move=="exit"){
        cout<<"The game is exit. "<<endl;
    }

    board.movePiece(from, to);
    moveHistory.addMove(from, to);

    // Check for winning condition
    if (isLastRowEmpty(board, isPlayer1Turn)) {
        cout << (isPlayer1Turn ? playerName1 : playerName2) << " wins!" << endl;
        break;
    }

    isPlayer1Turn = !isPlayer1Turn; // Switch turns
}

}
```

- There is also a function that first add the move of player.



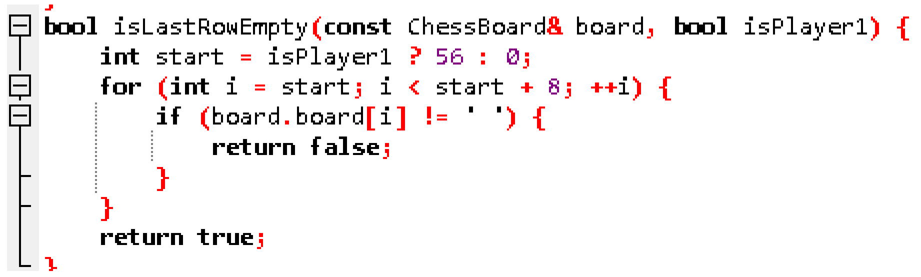
```
void addMove(int from, int to) {
    MoveNode* newMove = new MoveNode(from, to);
    newMove->next = head;
    head = newMove;
}
```

- There is a function to display the moves of the players.



```
void displayMoves() {
    MoveNode* current = head;
    while (current!=NULL) {
        cout << "Move from " << current->from << " to " << current->to << endl;
        current= current->next;
    }
}
```

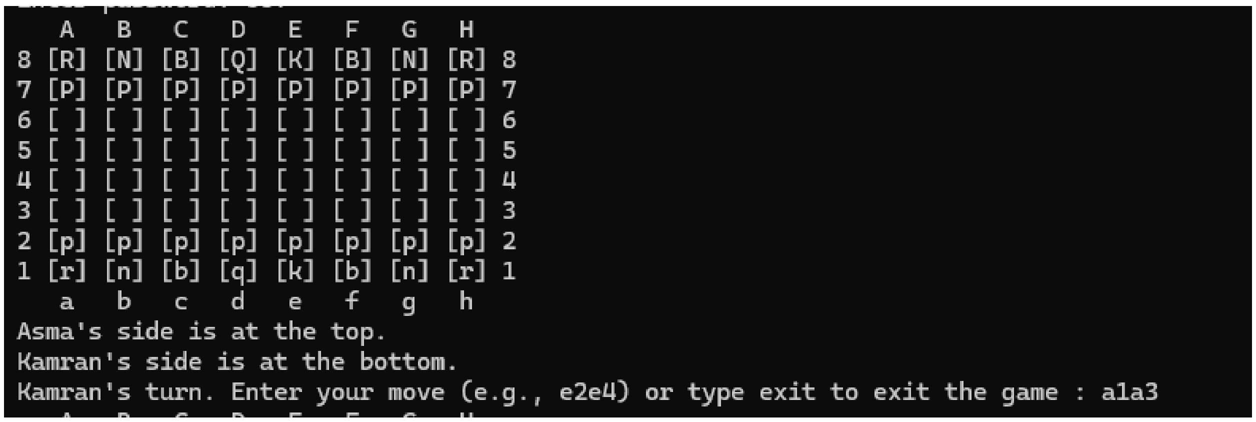
❖ The code also has a winning condition, that checks which player will win the game;



```
bool isLastRowEmpty(const ChessBoard& board, bool isPlayer1) {
    int start = isPlayer1 ? 56 : 0;
    for (int i = start; i < start + 8; ++i) {
        if (board.board[i] != ' ') {
            return false;
        }
    }
    return true;
}
```

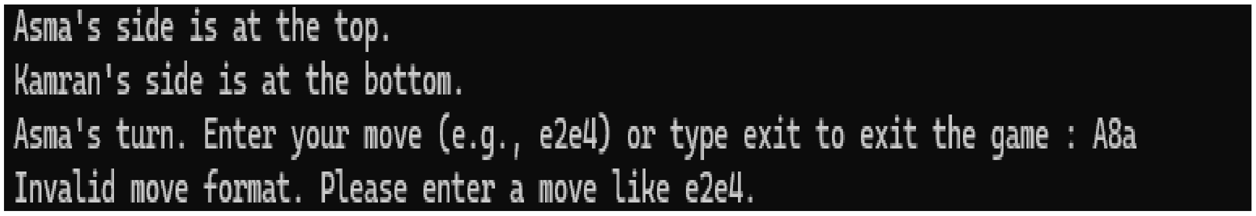
Output:

- When we start playing with the computer, the game will started. First I will do my move by using lower elements of chessboard and then player do its move by using upper elements of chessboard.



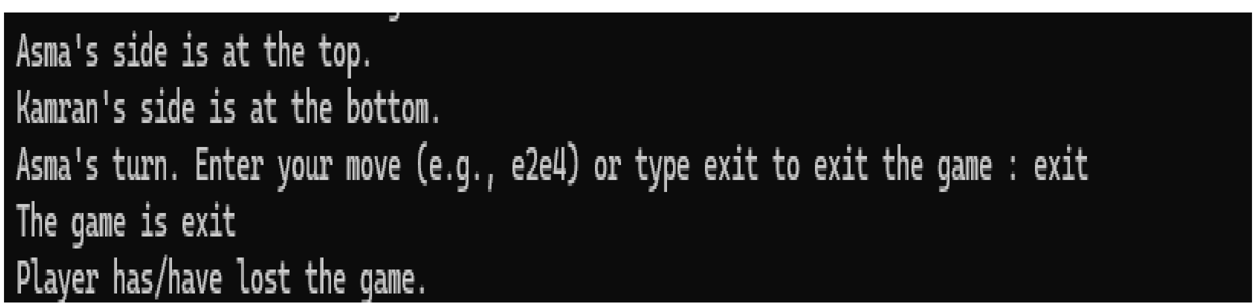
```
   A   B   C   D   E   F   G   H
8 [R] [N] [B] [Q] [K] [B] [N] [R] 8
7 [P] [P] [P] [P] [P] [P] [P] [P] 7
6 [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] 6
5 [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] 5
4 [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] 4
3 [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] 3
2 [p] [p] [p] [p] [p] [p] [p] [p] 2
1 [r] [n] [b] [q] [k] [b] [n] [r] 1
   a   b   c   d   e   f   g   h
Asma's side is at the top.
Kamran's side is at the bottom.
Kamran's turn. Enter your move (e.g., e2e4) or type exit to exit the game : a1a3
```

- If any person do invalid move i.e do not follow the move instructions of game, the game will show the “invalid move” and ask for the move again.

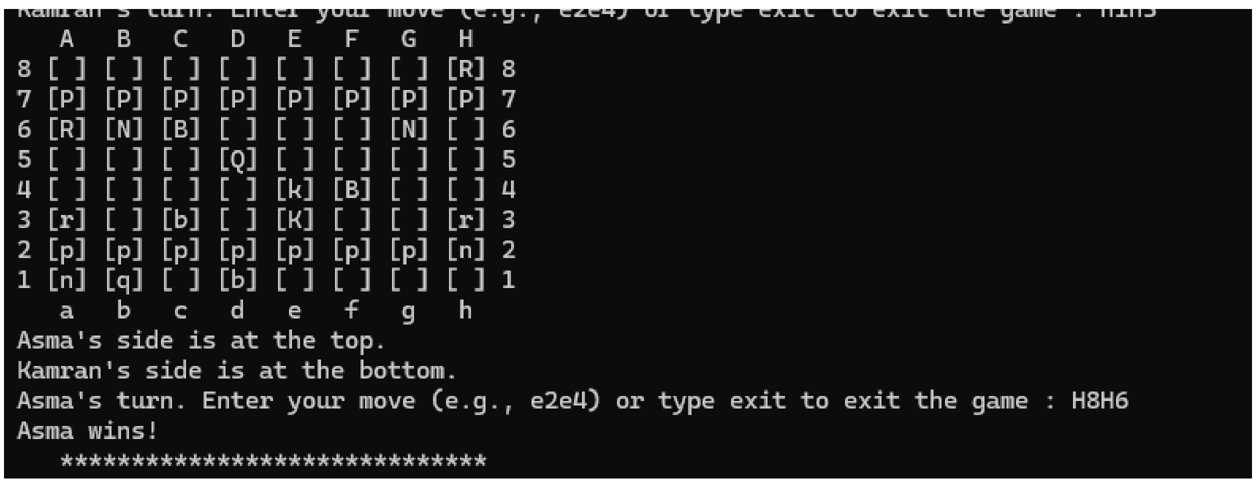


```
Asma's side is at the top.
Kamran's side is at the bottom.
Asma's turn. Enter your move (e.g., e2e4) or type exit to exit the game : A8a
Invalid move format. Please enter a move like e2e4.
```

- The game will exit if any of the player type exit. The player who will type “exit” will loss the game because he has left the game.


```
Asma's side is at the top.
Kamran's side is at the bottom.
Asma's turn. Enter your move (e.g., e2e4) or type exit to exit the game : exit
The game is exit
Player has/have lost the game.
```

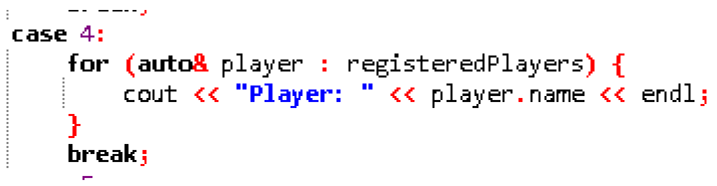
- I have made the winning condition in this game that the player whose last row will be emptied first will won the game;


```
   A   B   C   D   E   F   G   H
8 [ ] [ ] [ ] [ ] [ ] [ ] [ ] [R] 8
7 [P] [P] [P] [P] [P] [P] [P] [P] 7
6 [R] [N] [B] [ ] [ ] [ ] [N] [ ] 6
5 [ ] [ ] [ ] [Q] [ ] [ ] [ ] [ ] 5
4 [ ] [ ] [ ] [ ] [k] [B] [ ] [ ] 4
3 [r] [ ] [b] [ ] [K] [ ] [ ] [r] 3
2 [p] [p] [p] [p] [p] [p] [p] [n] 2
1 [n] [q] [ ] [b] [ ] [ ] [ ] [ ] 1
   a   b   c   d   e   f   g   h
Asma's side is at the top.
Kamran's side is at the bottom.
Asma's turn. Enter your move (e.g., e2e4) or type exit to exit the game : H8H6
Asma wins!
   ******************************
```

You have seen that Asma has won the game because she emptied the last row first.

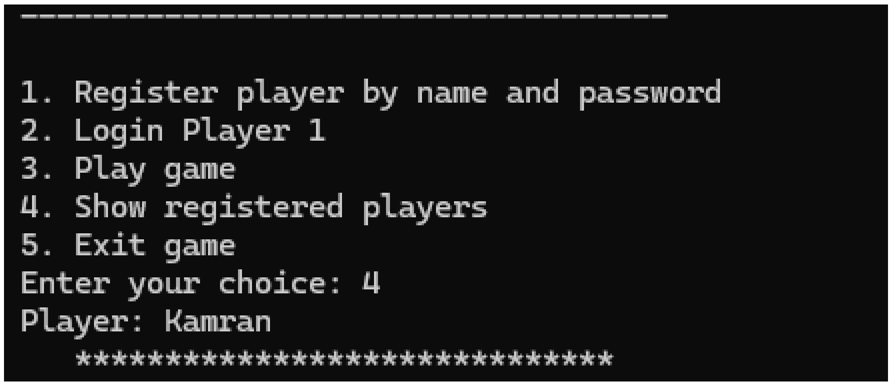
❖When we press 4, it shows the players that are registered in the game.

In this code, I have started the for loop in which I call registered player function. It means that loop will runs until all the registered players will not be shown.


```
case 4:
    for (auto& player : registeredPlayers) {
        cout << "Player: " << player.name << endl;
    }
    break;
```

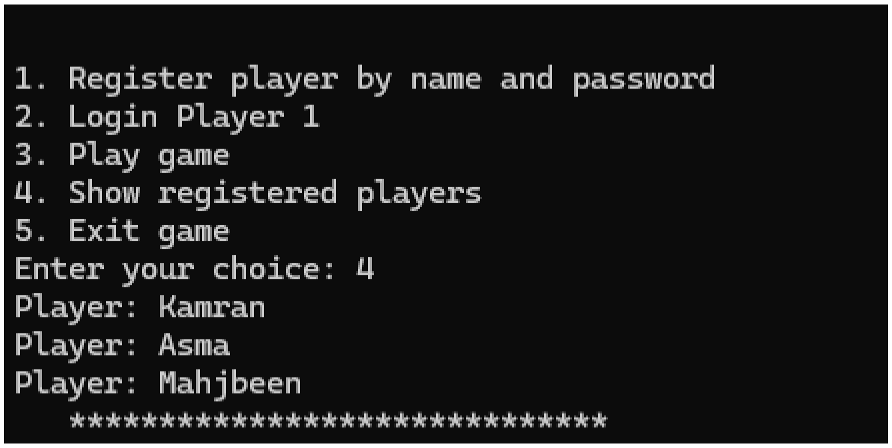
Output:

- When only player has login ;



```
------------------------------------

1. Register player by name and password
2. Login Player 1
3. Play game
4. Show registered players
5. Exit game
Enter your choice: 4
Player: Kamran
   ******************************
```

- When two or players have registered;

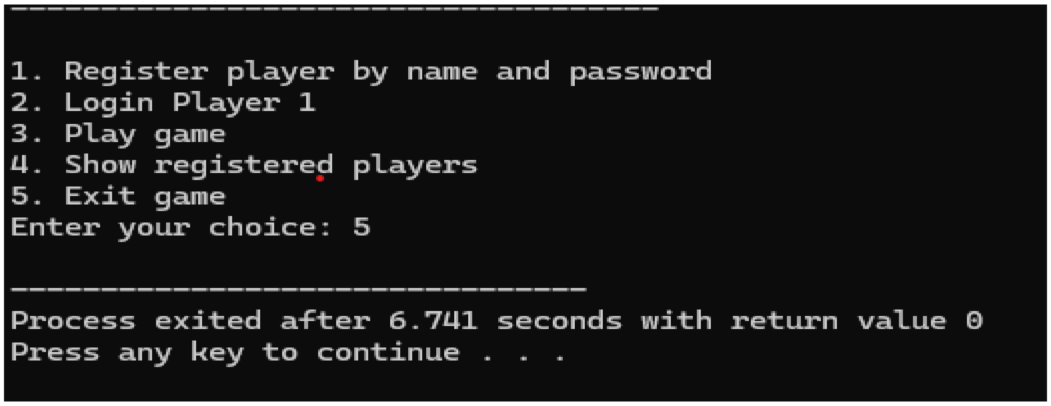


```

1. Register player by name and password
2. Login Player 1
3. Play game
4. Show registered players
5. Exit game
Enter your choice: 4
Player: Kamran
Player: Asma
Player: Mahjbeen
   ******************************
```

❖ When we press 5, it will exit the game and loop ends;

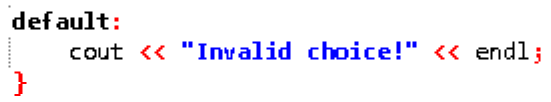
```
        break;
    case 5:
        exit(0);
        break;
```

Output:



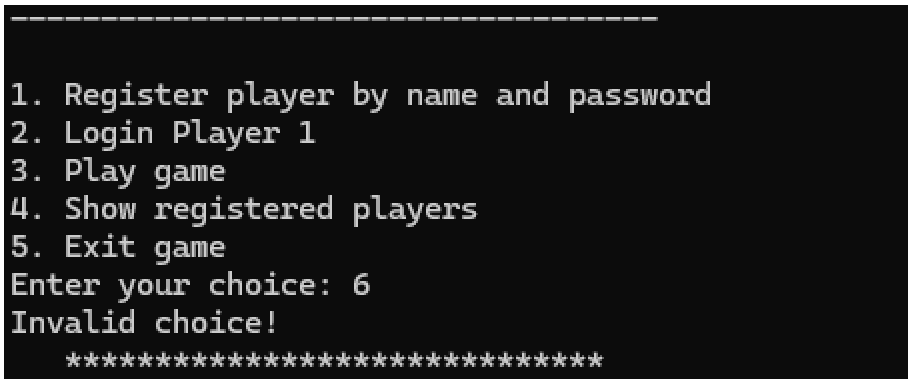
```
------------------------------------

1. Register player by name and password
2. Login Player 1
3. Play game
4. Show registered players
5. Exit game
Enter your choice: 5

--------------------------------
Process exited after 6.741 seconds with return value 0
Press any key to continue . . .
```

- **When we press any other key from these 5, it shows the invalid choice;**



```
default:
    cout << "Invalid choice!" << endl;
}
```

Output:



```
------------------------------------

1. Register player by name and password
2. Login Player 1
3. Play game
4. Show registered players
5. Exit game
Enter your choice: 6
Invalid choice!
   ******************************
```

FINAL CODE OF CHESS GAME

CODE:

```
#include <iostream>

#include <stack>

#include <queue>

#include <vector>

#include <string>

#include <cstdlib>  // for random computer move

#include <ctime>    // for random seed

using namespace std;

// Player class to store player information (name and password)

class Player {

public:

   string name;

   string password;

Player(string playerName, string playerPassword) {

           name=playerName;
password=playerPassword;    }     };

// Chessboard class (One-Dimensional Array)

class ChessBoard {

public:

    char board[64]; // 8x8 chessboard in 1D array

    ChessBoard() {

        for (int i = 0; i < 64; ++i) {

            board[i] = ' '; // Initialize empty spaces       }

        board[0] = 'R'; board[1] = 'N'; board[2] = 'B'; board[3] = 'Q'; board[4] = 'K'; board[5] = 'B'; board[6] =
'N'; board[7] = 'R';

        board[8] = 'P'; board[9] = 'P'; board[10] = 'P'; board[11] = 'P'; board[12] = 'P'; board[13] = 'P';
board[14] = 'P'; board[15] = 'P';

        board[48] = 'p'; board[49] = 'p'; board[50] = 'p'; board[51] = 'p'; board[52] = 'p'; board[53] = 'p';
board[54] = 'p'; board[55] = 'p';

        board[56] = 'r'; board[57] = 'n'; board[58] = 'b'; board[59] = 'q'; board[60] = 'k'; board[61] = 'b';
board[62] = 'n'; board[63] = 'r';        }


    void displayBoard(bool isComputerOpponent, string player1, string player2) {

        cout << "   A   B   C   D   E   F   G   H" << endl;

        for (int i = 0; i < 8; ++i) {

            cout << 8 - i << " "; // Row labels (reverse order for user side at the bottom)

            for (int j = 0; j < 8; ++j) {

                cout << "[" << board[i * 8 + j] << "] "; // Print board piece with brackets for better clarity       }

            cout << 8 - i << endl; // Row labels       }

        cout << "   a   b   c   d   e   f   g   h" << endl;

        cout << player2 << "'s side is at the top." << endl;

        cout << player1 << "'s side is at the bottom." << endl;      }

void movePiece(int from, int to) {

        // Basic move logic (to be expanded)

        board[to] = board[from];

        board[from] = ' ';    }
```

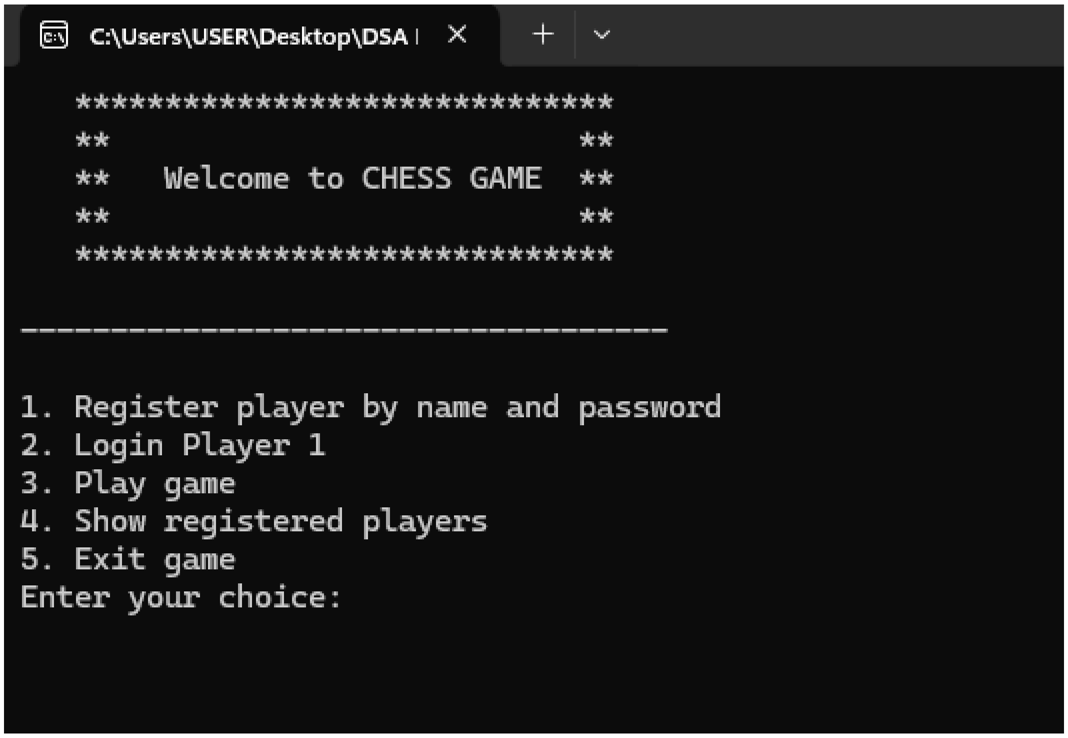
```
     int getPositionFromInput(string move) {
       // Convert input like e2e4 to a position index in the board
       int fromCol = tolower(move[0]) - 'a'; // Convert to lowercase for uniformity
       int fromRow = 8 - (move[1] - '0');
       int toCol = tolower(move[2]) - 'a'; // Convert to lowercase for uniformity
       int toRow = 8 - (move[3] - '0');
       return fromRow * 8 + fromCol;   }
    void computerMove() {
       // Simple random move logic for the computer (to be expanded)
       srand(time(0));
       int from, to;
       do {     from = rand() % 64;
                 to = rand() % 64;     }
 while (board[from] == ' ' || islower(board[from]) || (board[to] != ' ' && islower(board[to]))); // Ensure
valid move
 movePiece(from, to);
 cout << "Computer move: " << char('A' + (to % 8)) << (8 - (to / 8)) << endl; // Display computer move }   };
// Node for single linked list (Move history)
struct MoveNode {
    int from, to;
    MoveNode* next;
    MoveNode(int f, int t) {      from=f;
                                   to=t;
                                   next=NULL;    }    };
// Single Linked List for storing moves
class MoveHistory {
public:
    MoveNode* head;
    MoveHistory() {
            head=NULL;
            }
    void addMove(int from, int to) {
        MoveNode* newMove = new MoveNode(from, to);
        newMove->next = head;
        head = newMove;   }
void displayMoves() {
        MoveNode* current = head;
        while (current!=NULL) {
            cout << "Move from " << current->from << " to " << current->to << endl;
            current= current->next;     }     }     };
// Node for circular linked list (Turn-based player switching)
struct PlayerNode {
    string playerName;
    PlayerNode* next;
    PlayerNode(string name){
              playerName=name;
              next=NULL;   }    };
// Circular Linked List for player turns
class PlayerTurnQueue {
public:
    PlayerNode* head;
    PlayerNode* tail;
   PlayerTurnQueue(){     head=NULL;
                                        tail=NULL;    }
  void addPlayer(string playerName) {
        PlayerNode* newPlayer = new PlayerNode(playerName);
        if (!head) {
            head = tail = newPlayer;
            newPlayer->next = head;  // Circular link     } else {
            tail->next = newPlayer;
            tail = newPlayer;
            tail->next = head;  // Circular link       }     }
   void nextTurn() {
        if (head) {
            cout << "It's " << head->playerName << "'s turn." << endl;
            head = head->next;    }     }      };
// Node for double linked list (Piece history for undo/redo)
struct PieceNode {
    char piece;
    int position;
    PieceNode* prev;
    PieceNode* next;
    PieceNode(char p, int pos) {
             piece=p;
             position=pos;
             prev=NULL;
             next=NULL;    }    };
// Double Linked List for piece history
class PieceHistory {
public:
    PieceNode* head;
    PieceNode* tail;
 PieceHistory(){
             head=NULL;
             tail=NULL;   }
void addPiece(char piece, int position) {
        PieceNode* newPiece = new PieceNode(piece, position);
        if (!head) {
            head = tail = newPiece;
        } else {
            tail->next = newPiece;
            newPiece->prev = tail;
            tail = newPiece;   }    }
void undoMove() {
        if (tail) {
            cout << "Undo move: " << tail->piece << " from position " << tail->position << endl;
            PieceNode* temp = tail;
            tail = tail->prev;
            if (tail) tail->next = nullptr;
            delete temp;    }     }
  void redoMove() {
        if (head) {
            cout << "Redo move: " << head->piece << " to position " << head->position << endl;
            PieceNode* temp = head;
            head = head->next;
            if (head) head->prev = nullptr;
            delete temp;      }    }    };
// Stack for undo game states
class GameStateStack {
public:
    stack<ChessBoard> gameStateHistory;
  void saveState(ChessBoard& board) {
        gameStateHistory.push(board);    }
void undoState(ChessBoard& board) {
        if (!gameStateHistory.empty()) {
            board = gameStateHistory.top();
            gameStateHistory.pop();
            cout << "Game state reverted!" << endl;
        } else {
            cout << "No previous game state to undo." << endl;     }     }    };
```

```
// Queue for managing pending moves
class MoveQueue {
public:
    queue<string> moveQueue;
void addMove(string move) {
        moveQueue.push(move);  }
  void executeMove() {
        if (!moveQueue.empty()) {
            cout << "Executing move: " << moveQueue.front() << endl;
            moveQueue.pop();   }   }   };
// Global list of registered players
vector<Player> registeredPlayers;
Player* loggedInPlayer = nullptr;
//Player* loggedInPlayer2 = nullptr;
// Function to display the main menu
void displayMainMenu() {
              cout<<"   ******************************"<<endl;
              cout<<"   **                         **"<<endl;
              cout<<"   **   Welcome to CHESS GAME  ** "<<endl;
              cout<<"   **                         ** "<<endl;
              cout<<"   ******************************"<<endl;
              cout<<" \n----------------------------------- "<<endl;
    cout << "\n1. Register player by name and password" << endl;
    cout << "2. Login Player 1" << endl;
    //cout << "3. Login Player 2" << endl;
    cout << "3. Play game" << endl;
    cout << "4. Show registered players" << endl;
    cout << "5. Exit game" << endl;
    cout << "Enter your choice: ";
}
```

```
// Function to register a player
void registerPlayer() {
    string name, password;
    cout << "Enter player name: ";   cin >> name;
    cout << "Enter player password: ";  cin >> password;
 Player newPlayer(name, password);
    registeredPlayers.push_back(newPlayer);
    cout << "Player " << name << " registered successfully!" << endl;   }
// Function to login a player
void loginPlayer() {
    string name, password;
    cout << "Enter player name: ";   cin >> name;
    cout << "Enter player password: ";   cin >> password;
    bool found = false;
    for (auto& player : registeredPlayers) {
        if (player.name == name && player.password == password) {
            loggedInPlayer = &player;
            found = true;
            cout << "Login successful!" << endl;    break;   }   }
 if (!found) {      cout << "Invalid credentials!" << endl;    }     }
// Function to make a random move for the computer
void computerMove(ChessBoard& board) {
    // Simple random move for the computer (for demonstration purposes)
    int from = rand() % 64;
    int to = rand() % 64;
    board.movePiece(from, to);
    cout << "Computer moves from " << from << " to " << to << endl;   }
bool isLastRowEmpty(const ChessBoard& board, bool isPlayer1) {
    int start = isPlayer1 ? 56 : 0;
    for (int i = start; i < start + 8; ++i) {
        if (board.board[i] != ' ') {     return false;    }        }      return true;    }
// Function to play the game
void playGame() {
              string playerMove;
 ChessBoard board;
    MoveHistory moveHistory;
    PlayerTurnQueue turnQueue;
    PieceHistory pieceHistory;
    GameStateStack gameStateStack;
    MoveQueue moveQueue;
 string playerName1, playerName2;
    cout << "Enter Player 1 name: ";   cin >> playerName1;
    cout << "Enter Player 2 name (uppercase moves): ";   cin >> playerName2;
    int gameChoice;
    cout << "1. Play with computer" << endl;
    cout << "2. Play with another player" << endl;
    cout << "Enter your choice: ";   cin >> gameChoice;
 if (gameChoice == 1) {
        turnQueue.addPlayer(playerName1);
        turnQueue.addPlayer("Computer");
 bool gameOver = false;
bool isPlayer1Turn = true;
      while (true) {
        board.displayBoard(false, playerName1,"Computer");
        string move;
        cout << (isPlayer1Turn ? playerName1 : "Computer") << "'s turn. Enter your move (e.g., e2e4) or
type exit to exit the game : ";
        cin >> move;
 if (move.length() != 4) {
            cout << "Invalid move format. Please enter a move like e2e4." << endl;
            continue;   }
```

```
else if(move=="exit"){
              cout<<"The game is exit. "<<endl;  }
int from = board.getPositionFromInput(move.substr(0, 2));
 int to = board.getPositionFromInput(move.substr(2, 2));
   if (from < 0 || from >= 64 || to < 0 || to >= 64 || board.board[from] == ' ') {
            cout << "Player has lost the game. " << endl;    break;   }
                         else if(move=="exit"){
              cout<<"The game is exit "<<endl;  }
board.movePiece(from, to);
board.displayBoard(true, playerName1, "Computer");
   // Check for winning condition
        if (isLastRowEmpty(board, isPlayer1Turn)) {
            cout << (isPlayer1Turn ? playerName1 : "Computer") << " wins!" << endl;
            break;   }
    board.computerMove();       }
    } else if (gameChoice == 2) {
        if (registeredPlayers.size() < 2) {
            cout << "Not enough players are logged in. Please log in the second player." << endl;
            string playerName2, password2;
            cout << "Enter second player name: ";  cin >> playerName2;
            cout << "Enter password: ";  cin >> password2;
 Player secondPlayer(playerName2, password2);
            registeredPlayers.push_back(secondPlayer);   }
 playerName2 = registeredPlayers[1].name;
        turnQueue.addPlayer(playerName1);
        turnQueue.addPlayer(playerName2);
     bool isPlayer1Turn = true;
      while (true) {
        board.displayBoard(false, playerName1,playerName2);
        string move;
        cout << (isPlayer1Turn ? playerName1 : playerName2) << "'s turn. Enter your move (e.g., e2e4) or
type exit to exit the game : ";    cin >> move;
  if (move.length() != 4) {
            cout << "Invalid move format. Please enter a move like e2e4." << endl;
            continue;  }
        else if(move=="exit"){
              cout<<"The game is exit "<<endl;  }
 int from = board.getPositionFromInput(move.substr(0, 2));
 int to = board.getPositionFromInput(move.substr(2, 2));
if (from < 0 || from >= 64 || to < 0 || to >= 64 || board.board[from] == ' ') {
            cout << "Player has/have lost the game." << endl;
            break;   }
              else if(move=="exit"){
              cout<<"The game is exit. "<<endl;  }
 board.movePiece(from, to);
 moveHistory.addMove(from, to);
  // Check for winning condition
        if (isLastRowEmpty(board, isPlayer1Turn)) {
            cout << (isPlayer1Turn ? playerName1 : playerName2) << " wins!" << endl;
            break;  }
 isPlayer1Turn = !isPlayer1Turn; // Switch turns  }   } }
int main() {
    srand(time(0));  // Seed the random number generator for computer moves
  int choice;
while (true) {
        displayMainMenu();
        cin >> choice;
 switch (choice) {
        case 1:
            registerPlayer();
            break;
        case 2:
          loginPlayer();   break;
        case 3:
          playGame();     break;
        case 4:
          for (auto& player : registeredPlayers) {
            cout << "Players that have login :  " << player.name << endl;   }    break;
        case 5:
          exit(0);    break;
        default:
          cout << "Invalid choice!" << endl;    }    }
return 0;    }
```

Final Output:



C:\Users\USER\Desktop\DSA

```
   ******************************
   **                          **
   **   Welcome to CHESS GAME  **
   **                          **
   ******************************

------------------------------------

1. Register player by name and password
2. Login Player 1
3. Play game
4. Show registered players
5. Exit game
Enter your choice:
```

THE END