**Name: Muhammad Anas Baig**

**Enrollment No.: 01-134152-037**

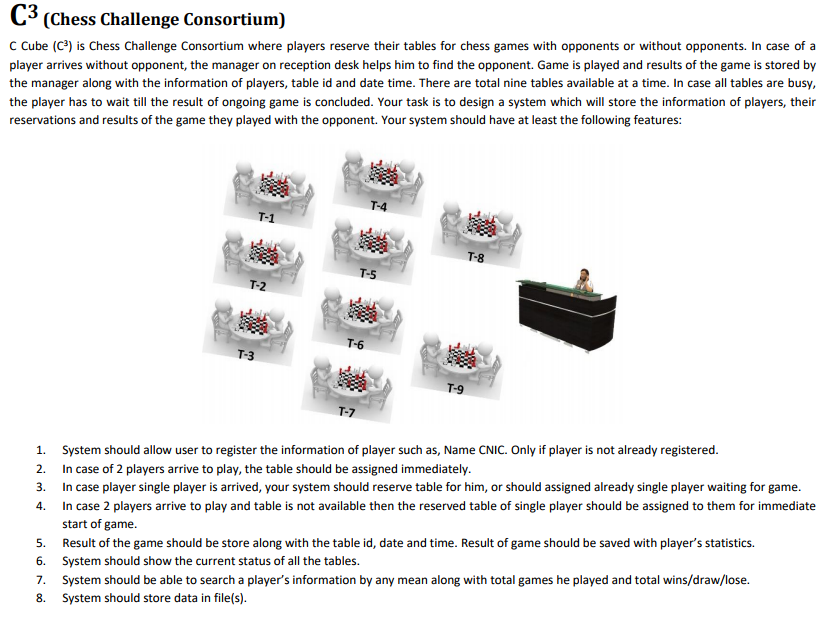
**Section: BS(CS)-5A**

**ASSIGNMENT # 1**

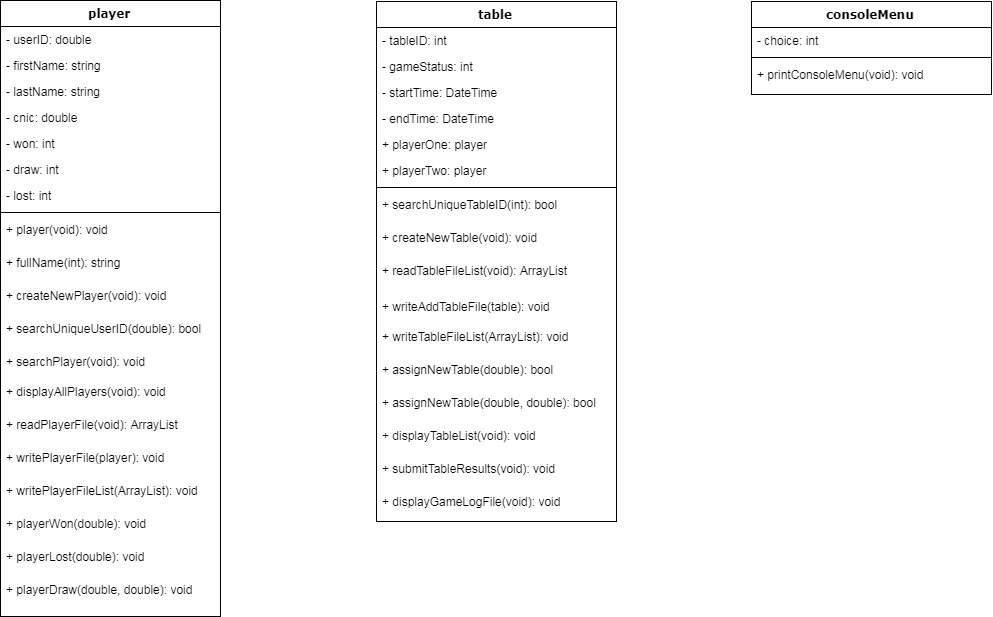
**Visual Programming**

**BS(CS) – 5A**

**October 8, 2017**



**UML CLASS DIAGRAM:**

****

**Source Code:**

**consoleMenu.cs File:**

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
6. **namespace** Assignment1
7. {
8. **class** consoleMenu
9. {
10. **public** **void** printConsoleMenu() //method to print console menu
11. {
12. **int** choice; //variable to store user choices
13. **do** //loop to display after every operation
14. {
15. Console.Write("--------------------------------------------------------------------------------");
16. Console.WriteLine( "MENU:" );
17. Console.Write("--------------------------------------------------------------------------------");
18. Console.WriteLine("1. Register New Player.");
19. Console.WriteLine("2. Search Player.");
20. Console.WriteLine("3. Display All Players Statistics.");
21. Console.WriteLine("4. Assign Table to One Player.");
22. Console.WriteLine("5. Assign Table to Two Players.");
23. Console.WriteLine("6. Submit Table Results.");
24. Console.WriteLine("7. Display All Tables Status.");
25. Console.WriteLine("8. Add New Table to System.");
26. Console.WriteLine("9. Display Game Log History.");
27. Console.WriteLine("10. Exit." );
28. Console.Write("--------------------------------------------------------------------------------");
29. Console.WriteLine( "SELECT DESIRED OPERATION:" );
30. Console.Write("--------------------------------------------------------------------------------");
31. choice = **int**.Parse(Console.ReadLine());
33. **if** (choice == 1)
34. {
35. player p = **new** player();
36. p.createNewPlayer(); //to register new player in the system
37. }
38. **else** **if** (choice == 2)
39. {
40. player p = **new** player();
41. p.searchPlayer(); //to search player in the system
42. }
43. **else** **if** (choice == 3)
44. {
45. player p = **new** player();
46. p.displayAllPlayers(); //to display all players statistics
47. }
48. **else** **if** (choice == 4)
49. {
50. table t = **new** table();
51. Console.Write("--------------------------------------------------------------------------------");
52. Console.WriteLine("ASSIGN TABLE TO ONE PLAYER:");
53. Console.Write("--------------------------------------------------------------------------------");
54. player p = **new** player();
55. **int** userID;
56. **do** //checks either the userID assigning to the table exists or not
57. {
58. Console.WriteLine("Enter Player User-ID:");
59. userID = **int**.Parse(Console.ReadLine());
60. **if** (!p.searchUniqueUserID(userID)) //checks either the userID assigning to the table exists or not
61. {
62. Console.WriteLine("ERROR! User-ID not found, please try again.");
63. }
64. }
65. **while** (!p.searchUniqueUserID(userID)); //checks either the userID assigning to the table exists or not
66. **if** (!t.assignNewTable(userID)) //checks that all tables are filled or not, if table is partially filled then it will be assigned to that player
67. {
68. Console.WriteLine("PLEASE WAIT! All tables are filled.");
69. }
70. }
71. **else** **if** ( choice == 5 )
72. {
73. table t = **new** table();
74. Console.Write("--------------------------------------------------------------------------------");
75. Console.WriteLine("ASSIGN TABLE TO TWO PLAYERS:");
76. Console.Write("--------------------------------------------------------------------------------");
77. player p = **new** player();
78. **int** userOneID;
79. **int** userTwoID;
80. **do** //checks either the userID assigning to the table exists or not
81. {
82. Console.WriteLine("Enter Player-1 User-ID:");
83. userOneID = **int**.Parse(Console.ReadLine());
84. Console.WriteLine("Enter Player-2 User-ID:");
85. userTwoID = **int**.Parse(Console.ReadLine());
86. **if** (!p.searchUniqueUserID(userOneID) && !p.searchUniqueUserID(userTwoID)) //checks either the userID assigning to the table exists or not
87. {
88. Console.WriteLine("ERROR! User-ID not found, please try again.");
89. }
90. }
91. **while** (!p.searchUniqueUserID(userOneID) && !p.searchUniqueUserID(userTwoID)); //checks either the userID assigning to the table exists or not
92. **if** (!t.assignNewTable(userOneID, userTwoID)) //checks that all tables are filled or not and then assign table to players
93. {
94. Console.WriteLine("PLEASE WAIT! All tables are filled.");
95. }
96. }
97. **else** **if** (choice == 6)
98. {
99. table t = **new** table();
100. t.submitTableResults(); //to submit results of the table
101. }
102. **else** **if** (choice == 7)
103. {
104. table t = **new** table();
105. t.displayTableList(); //to display all tables status
106. }
107. **else** **if** (choice == 8)
108. {
109. table t = **new** table();
110. t.createNewTable(); //to add new table to system
111. }
112. **else** **if** (choice == 9)
113. {
114. table t = **new** table();
115. t.displayGameLogFile(); //to display game history
116. }
117. **else**
118. {
119. Console.WriteLine("ERROR! Invalid Input.");
120. }
121. }
122. **while** (choice != 10); //loop to display after every operation
123. }
124. }
125. }

**player.cs File:**

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.IO;
6. **using** System.Collections;
8. **namespace** Assignment1
9. {
10. **class** player
11. {
12. **double** userID;
13. **string** firstName;
14. **string** lastName;
15. **double** cnic;
16. **int** won;
17. **int** draw;
18. **int** lost;
20. **public** **double** userIDProperty
21. {
22. **get**{    **return** userID;  }
23. **set**{    userID = value; }
24. }
25. **public** **string** firstNameProperty
26. {
27. **get**{    **return** firstName;   }
28. **set**{    firstName = value;  }
29. }
30. **public** **string** lastNameProperty
31. {
32. **get**{    **return** lastName;    }
33. **set**{    lastName = value;   }
34. }
35. **public** **double** cnicProperty
36. {
37. **get**{    **return** cnic;    }
38. **set**{    cnic = value;   }
39. }
40. **public** **int** wonProperty
41. {
42. **get**{    **return** won;     }
43. **set**{    won = value;    }
44. }
45. **public** **int** drawProperty
46. {
47. **get** {   **return** draw;    }
48. **set** {   draw = value;   }
49. }
50. **public** **int** lostProperty
51. {
52. **get**{    **return** lost;    }
53. **set**{    lost = value;   }
54. }
55. **public** player()
56. {
57. won = 0;
58. draw = 0;
59. lost = 0;
60. }
61. **public** **string** fullName(**double** id) //function that returns concatenation of firstname and lastname
62. {
63. ArrayList playerList = **new** ArrayList(); //player list
64. playerList = readPlayerFile(); //reading file to list
66. **for** (**int** i = 0; i < playerList.Count; i++) //checks each user in list
67. {
68. **if** ((playerList[i] **as** player).userID == id) //checks for the required user
69. {
70. **return** ((playerList[i] **as** player).firstName + " " + (playerList[i] **as** player).lastName); //returns concatenation of firstname and lastname
71. }
72. }
73. **return** (""); //if user not found
74. }
75. **public** **void** createNewPlayer()
76. {
77. Console.Write("--------------------------------------------------------------------------------");
78. Console.WriteLine( "CREATE NEW PLAYER:" );
79. Console.Write("--------------------------------------------------------------------------------");
80. **do** //loop that prompts user to enter new userID if the typed one is already taken
81. {
82. Console.WriteLine("Enter New User-ID:");
83. **this**.userIDProperty = **double**.Parse(Console.ReadLine());
84. **if** ( searchUniqueUserID(**this**.userIDProperty) )
85. {
86. Console.WriteLine( "ERROR! User-ID already assigned, kindly choose another.");
87. }
88. }
89. **while** (searchUniqueUserID(**this**.userIDProperty)); //loop that prompts user to enter new userID if the typed one is already taken
90. Console.WriteLine("Enter First Name:");
91. **this**.firstNameProperty = Console.ReadLine();
92. Console.WriteLine("Enter Last Name:");
93. **this**.lastNameProperty = Console.ReadLine();
94. Console.WriteLine("Enter CNIC:");
95. **this**.cnicProperty = **double**.Parse(Console.ReadLine());
96. Console.WriteLine("Player Succesfully Registered");
98. writePlayerFile(**this**); //appends the new player in player file
99. }
100. **public** **bool** searchUniqueUserID(**double** id) //before creating new user this methods checks either the userID is already taken or not
101. {
102. ArrayList playerList = **new** ArrayList(); //player list
103. playerList = readPlayerFile(); //reading file to list
105. **for** (**int** i = 0; i < playerList.Count; i++)  //checks each user in list
106. {
107. **if** ((playerList[i] **as** player).userID == id) //checks either userID is already taken or not
108. {
109. **return** **true**;
110. }
111. }
112. **return** **false**;
113. }
114. **public** **void** searchPlayer() //to search a specific user in the system
115. {
116. ArrayList playerList = **new** ArrayList(); //player list
117. playerList = readPlayerFile(); //reading file to list
119. Console.Write( "--------------------------------------------------------------------------------" );
120. Console.WriteLine("SEARCH PLAYER:");
121. Console.Write("--------------------------------------------------------------------------------");
122. Console.WriteLine("Enter Desired Operation:");
123. Console.WriteLine("1. Search by User-ID.");
124. Console.WriteLine("2. Search by Name.");
125. Console.WriteLine("3. Search by CNIC.");
126. **int** choice = **int**.Parse(Console.ReadLine());
128. **if**(choice == 1) //search by userID
129. {
130. Console.WriteLine();
131. Console.WriteLine("Enter Search Player User-ID.");
132. **double** ID = **double**.Parse(Console.ReadLine());
134. **for** (**int** i = 0; i < playerList.Count; i++) //checks each user in list
135. {
136. **if** ((playerList[i] **as** player).userID == ID) //checks for the requied userID
137. {
138. Console.WriteLine("----------------------");
139. Console.WriteLine("Player - " + (i + 1) + " Statistics:");
140. Console.WriteLine("----------------------");
141. Console.WriteLine("User ID:     " + (playerList[i] **as** player).userID);
142. Console.WriteLine("Name:        " + (playerList[i] **as** player).firstName + " " + (playerList[i] **as** player).lastName);
143. Console.WriteLine("CNIC:        " + (playerList[i] **as** player).cnic);
144. **int** gamesPlayed = ((playerList[i] **as** player).won + (playerList[i] **as** player).draw + (playerList[i] **as** player).lost); //number of games played
145. Console.WriteLine("SCORE:");
146. Console.WriteLine("------");
147. Console.WriteLine("Won:         " + (playerList[i] **as** player).won);
148. Console.WriteLine("Draw:        " + (playerList[i] **as** player).draw);
149. Console.WriteLine("Lost:        " + (playerList[i] **as** player).lost);
150. Console.WriteLine("----------------------");
151. Console.WriteLine("Total Games: " + gamesPlayed);
152. Console.WriteLine("----------------------");
153. }
154. }
155. }
156. **else** **if**(choice == 2) //search by name
157. {
158. Console.WriteLine("Enter Search Player Full Name:");
159. **string** name = Console.ReadLine();
161. **for** (**int** i = 0; i < playerList.Count; i++) //checks each user in list
162. {
163. **if** (((playerList[i] **as** player).firstName + " " + (playerList[i] **as** player).lastName) == name) //checks for the requied name(firstName + lastName)
164. {
165. Console.WriteLine("----------------------");
166. Console.WriteLine("Player - " + (i + 1) + " Statistics:");
167. Console.WriteLine("----------------------");
168. Console.WriteLine("User ID:     " + (playerList[i] **as** player).userID);
169. Console.WriteLine("Name:        " + (playerList[i] **as** player).firstName + " " + (playerList[i] **as** player).lastName);
170. Console.WriteLine("CNIC:        " + (playerList[i] **as** player).cnic);
171. **int** gamesPlayed = ((playerList[i] **as** player).won + (playerList[i] **as** player).draw + (playerList[i] **as** player).lost); //number of games played
172. Console.WriteLine( "SCORE:");
173. Console.WriteLine("------");
174. Console.WriteLine("Won:         " + (playerList[i] **as** player).won);
175. Console.WriteLine("Draw:        " + (playerList[i] **as** player).draw);
176. Console.WriteLine("Lost:        " + (playerList[i] **as** player).lost);
177. Console.WriteLine("----------------------" );
178. Console.WriteLine("Total Games: " + gamesPlayed);
179. Console.WriteLine("----------------------");
180. }
181. }
182. }
183. **else** **if**(choice == 3) //search by cnic
184. {
185. Console.WriteLine("Enter Search Player CNIC.");
186. **double** num = **double**.Parse(Console.ReadLine());
188. **for** (**int** i = 0; i < playerList.Count; i++) //checks each user in list
189. {
190. **if** ((playerList[i] **as** player).cnic == num) //check for the required cnic
191. {
192. Console.WriteLine("----------------------");
193. Console.WriteLine("Player - " + (i + 1) + " Statistics:");
194. Console.WriteLine("----------------------" );
195. Console.WriteLine("User ID:     " + (playerList[i] **as** player).userID);
196. Console.WriteLine("Name:        " + (playerList[i] **as** player).firstName + " " + (playerList[i] **as** player).lastName);
197. Console.WriteLine("CNIC:        " + (playerList[i] **as** player).cnic);
198. **int** gamesPlayed = (( playerList[i] **as** player).won + (playerList[i] **as** player).draw + (playerList[i] **as** player).lost); //number of games played
199. Console.WriteLine("SCORE:");
200. Console.WriteLine("------");
201. Console.WriteLine("Won:         " + (playerList[i] **as** player).won);
202. Console.WriteLine("Draw:        " + (playerList[i] **as** player).draw);
203. Console.WriteLine("Lost:        " + (playerList[i] **as** player).lost);
204. Console.WriteLine("----------------------");
205. Console.WriteLine("Total Games: " + gamesPlayed);
206. Console.WriteLine("----------------------");
207. }
208. }
209. }
210. **else**
211. {
212. Console.WriteLine("ERROR! Invalid Input.");
213. }
214. }
215. **public** **void** displayAllPlayers() //to display all players statistics
216. {
217. ArrayList playerList = **new** ArrayList(); //player list
218. playerList = readPlayerFile(); //reading file to list
219. Console.Write("--------------------------------------------------------------------------------");
220. Console.WriteLine( "DISPLAY ALL PLAYERS STATISTICS:");
221. Console.Write("--------------------------------------------------------------------------------");
223. **for** ( **int** i = 0; i < playerList.Count; i++ ) //checks each user in list
224. {
225. Console.WriteLine();
226. Console.WriteLine("----------------------");
227. Console.WriteLine("Player - " + (i+1) + " Statistics:");
228. Console.WriteLine("----------------------" );
229. Console.WriteLine("User ID:     " + (playerList[i] **as** player).userID);
230. Console.WriteLine("Name:        " + (playerList[i] **as** player).firstName + " " + (playerList[i] **as** player).lastName);
231. Console.WriteLine("CNIC:        " + (playerList[i] **as** player).cnic);
232. **int** gamesPlayed = ((playerList[i] **as** player).won + (playerList[i] **as** player).draw + (playerList[i] **as** player).lost); //number of games played
233. Console.WriteLine("SCORE:");
234. Console.WriteLine("------");
235. Console.WriteLine("Won:         " + (playerList[i] **as** player).won);
236. Console.WriteLine("Draw:        " + (playerList[i] **as** player).draw);
237. Console.WriteLine("Lost:        " + (playerList[i] **as** player).lost);
238. Console.WriteLine("----------------------");
239. Console.WriteLine("Total Games: " + gamesPlayed);
240. Console.WriteLine("----------------------");
241. }
242. }
243. **public** ArrayList readPlayerFile() //reads player file and returns ArrayList which containts all players data
244. {
245. ArrayList playerList = **new** ArrayList(); //to display all players statistics
246. StreamReader readPlayerFile = **new** StreamReader("Players.txt"); //reading file to list
247. player p;
249. **while** (!readPlayerFile.EndOfStream) //reading file till end
250. {
251. p = **new** player();
252. p.userID = **double**.Parse(readPlayerFile.ReadLine());
253. p.firstName = readPlayerFile.ReadLine();
254. p.lastName = readPlayerFile.ReadLine();
255. p.cnic = **double**.Parse(readPlayerFile.ReadLine());
256. p.won = **int**.Parse(readPlayerFile.ReadLine());
257. p.draw = **int**.Parse(readPlayerFile.ReadLine());
258. p.lost = **int**.Parse(readPlayerFile.ReadLine());
259. playerList.Add( p );
260. }
261. readPlayerFile.Close();
262. **return** playerList; //returning ArrayList which contains all players data
263. }
264. **public** **void** writePlayerFile(player p) //to add new player to the file
265. {
266. StreamWriter writePlayerFile = **new** StreamWriter("Players.txt", **true**); //appending the player file
268. writePlayerFile.WriteLine( p.userID );
269. writePlayerFile.WriteLine( p.firstName );
270. writePlayerFile.WriteLine( p.lastName );
271. writePlayerFile.WriteLine( p.cnic );
272. writePlayerFile.WriteLine( p.won );
273. writePlayerFile.WriteLine( p.draw );
274. writePlayerFile.WriteLine( p.lost );
276. writePlayerFile.Close();
277. }
278. **public** **void** writePlayerFileList(ArrayList playerList) //to write modified/updated data to file -> modify/update -> game Win/Loss
279. {
280. StreamWriter writePlayerFile = **new** StreamWriter("Players.txt"); //not opened in appended mode because all modified/updated data is to write to file -> modify/update -> game Win/Loss
282. **for** (**int** i = 0; i < playerList.Count; i++) //checks each user in list
283. {
284. writePlayerFile.WriteLine((playerList[i] **as** player ).userIDProperty);
285. writePlayerFile.WriteLine((playerList[i] **as** player).firstNameProperty);
286. writePlayerFile.WriteLine((playerList[i] **as** player).lastNameProperty);
287. writePlayerFile.WriteLine((playerList[i] **as** player).cnicProperty);
288. writePlayerFile.WriteLine((playerList[i] **as** player).wonProperty);
289. writePlayerFile.WriteLine((playerList[i] **as** player).drawProperty);
290. writePlayerFile.WriteLine((playerList[i] **as** player).lostProperty);
291. }
292. writePlayerFile.Close();
293. }
294. **public** **void** playerWon(**double** id) //takes userID and updates user's won games
295. {
296. ArrayList playerList = **new** ArrayList(); //player list
297. playerList = readPlayerFile(); //reading file to list
299. **for** (**int** i = 0; i < playerList.Count; i++) //checks each user in list
300. {
301. **if** ((playerList[i] **as** player).userIDProperty == id)
302. {
303. (playerList[i] **as** player).wonProperty = 1 + (playerList[i] **as** player).wonProperty; //increments in user's won games
304. }
305. }
306. writePlayerFileList(playerList);
307. }
308. **public** **void** playerLost(**double** id) //takes userID and updates user's won games
309. {
310. ArrayList playerList = **new** ArrayList(); //player list
311. playerList = readPlayerFile(); //reading file to list
313. **for** (**int** i = 0; i < playerList.Count; i++) //checks each user in list
314. {
315. **if** ((playerList[i] **as** player).userIDProperty == id) //checks if required user is found
316. {
317. (playerList[i] **as** player).lostProperty = 1 + (playerList[i] **as** player).lostProperty; //increments in user's won games
318. }
319. }
320. writePlayerFileList(playerList);
321. }
322. **public** **void** playerDraw(**double** id1, **double** id2) //takes userIDs of player1 and player2 and updates both user's draw games
323. {
324. ArrayList playerList = **new** ArrayList(); //player list
325. playerList = readPlayerFile(); //reading file to list
327. **for** (**int** i = 0; i < playerList.Count; i++) //checks each user in list
328. {
329. **if** ((playerList[i] **as** player).userIDProperty == id1)
330. {
331. (playerList[i] **as** player).drawProperty = 1 + (playerList[i] **as** player).drawProperty; //increments player's draw games
332. }
333. **if** ((playerList[i] **as** player).userIDProperty == id2)
334. {
335. (playerList[i] **as** player).drawProperty = 1 + (playerList[i] **as** player).drawProperty; //increments player's draw games
336. }
337. }
338. writePlayerFileList( playerList );
339. }
340. }
341. }

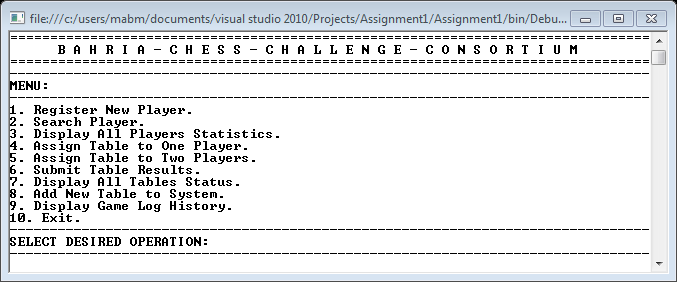
**table.cs File:**

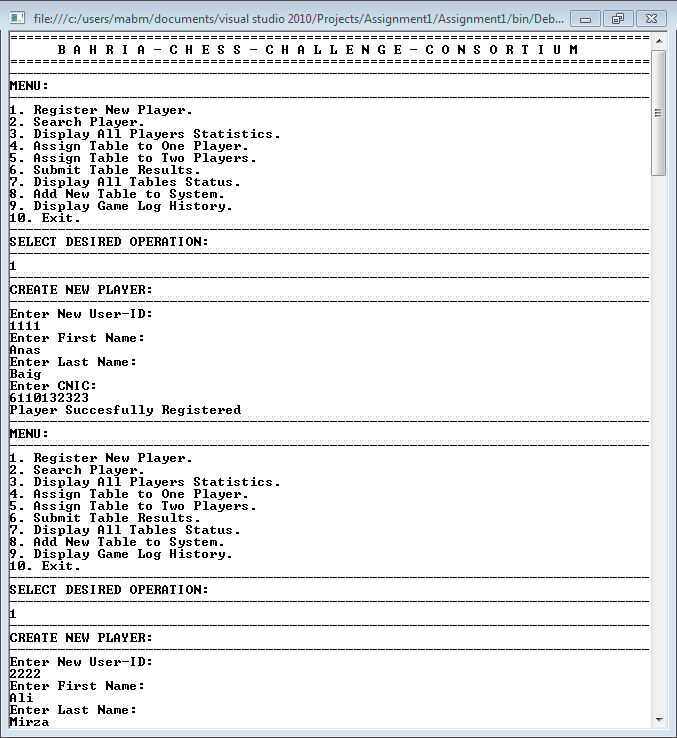
1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.IO;
6. **using** System.Collections;
8. **namespace** Assignment1
9. {
10. **class** table
11. {
12. **int** tableID; //stores tableID
13. **int** gameStatus; //stores game status i.e. 0->Empty Table, 1->One Player Assigned, 2->Two Players Assigned
14. DateTime startTime;
15. DateTime endTime;
16. **public** player playerOne = **new** player(); //player1 on table
17. **public** player playerTwo = **new** player(); //player2 on table
19. **public** **int** tableIDProperty
20. {
21. **get** { **return** tableID; }
22. **set** { tableID = value; }
23. }
24. **public** **int** gameStatusProperty
25. {
26. **get** { **return** gameStatus; }
27. **set** { gameStatus = value; }
28. }
29. **public** **double** playerOneProperty
30. {
31. **get** { **return** playerOne.userIDProperty; }
32. **set** { playerOne.userIDProperty = value; }
33. }
34. **public** **double** playerTwoProperty
35. {
36. **get** { **return** playerTwo.userIDProperty; }
37. **set** { playerTwo.userIDProperty = value; }
38. }
39. **public** DateTime startTimeProperty
40. {
41. **get** { **return** startTime; }
42. **set** { startTime = value; }
43. }
44. **public** DateTime endTimeProperty
45. {
46. **get** { **return** endTime; }
47. **set** { endTime = value; }
48. }
49. **public** **bool** searchUniqueTableID(**int** id) //while creating new table checks either the tableID in already assigned or not
50. {
51. ArrayList tableList = **new** ArrayList(); //ArrayList to store list of tables
52. tableList = readTableFileList(); //reads tables from file to list
54. **for** (**int** i = 0; i < tableList.Count; i++) //checks each table
55. {
56. **if** ((tableList[i] **as** table).tableID == id) //checks for the required tableID
57. {
58. **return** **true**;
59. }
60. }
61. **return** **false**;
62. }
63. **public** **void** createNewTable() //to add new table to system
64. {
65. Console.Write("--------------------------------------------------------------------------------");
66. Console.WriteLine("ADD NEW TABLE TO SYSTEM:");
67. Console.Write("--------------------------------------------------------------------------------");
69. **do**
70. {
71. Console.WriteLine("Enter New Table-ID:");
72. **this**.tableID = **int**.Parse(Console.ReadLine());
73. **if** (searchUniqueTableID(**this**.tableID)) //while creating new table checks either the tableID in already assigned or not
74. {
75. Console.WriteLine("ERROR! Table-ID already assigned, kindly choose another.");
76. }
77. }
78. **while** (searchUniqueTableID(**this**.tableID));
80. gameStatusProperty = 0; //game status i.e. 0->Empty Table, 1->One Player Assigned, 2->Two Players Assigned
81. playerOneProperty = 0;
82. playerTwoProperty = 0;
83. startTimeProperty = DateTime.Now;
84. endTimeProperty = DateTime.Now;
85. Console.WriteLine("Table Successfully Created.");
87. writeAddTableFile(**this**); //appends new to table to table file
88. }
89. **public** ArrayList readTableFileList() //reads table file to list and then returns list
90. {
91. ArrayList tableList = **new** ArrayList(); //ArrayList to store list of tables
92. StreamReader readTableFile = **new** StreamReader("Tables.txt"); //read file
93. table t;
95. **while** (!readTableFile.EndOfStream) //reads table file till end
96. {
97. t = **new** table();
98. t.tableIDProperty = **int**.Parse(readTableFile.ReadLine());
99. t.gameStatusProperty = **int**.Parse(readTableFile.ReadLine());
100. t.startTime = DateTime.Parse(readTableFile.ReadLine());
101. t.endTime = DateTime.Parse(readTableFile.ReadLine());
102. t.playerOneProperty = **double**.Parse(readTableFile.ReadLine());
103. t.playerTwoProperty = **double**.Parse(readTableFile.ReadLine());
104. tableList.Add(t);
105. }
106. readTableFile.Close();
107. **return** tableList;
108. }
109. **public** **void** writeAddTableFile(table t) //to add new table to the sysem by appending
110. {
111. StreamWriter writeTableFile = **new** StreamWriter("Tables.txt", **true**); //appending table file
113. writeTableFile.WriteLine(t.tableIDProperty);
114. writeTableFile.WriteLine(t.gameStatusProperty);
115. writeTableFile.WriteLine(t.startTimeProperty);
116. writeTableFile.WriteLine(t.endTimeProperty);
117. writeTableFile.WriteLine(t.playerOneProperty);
118. writeTableFile.WriteLine(t.playerTwoProperty);
120. writeTableFile.Close();
121. }
122. **public** **void** writeTableFileList(ArrayList tableList) //to write modified/updated data to file -> modify/update -> table status
123. {
124. StreamWriter writeTableFile = **new** StreamWriter("Tables.txt"); //not opened in appended mode because all modified/updated data is to write to file -> modify/update -> table status
126. **for** (**int** i = 0; i < tableList.Count; i++)
127. {
128. writeTableFile.WriteLine((tableList[i] **as** table).tableIDProperty);
129. writeTableFile.WriteLine((tableList[i] **as** table).gameStatusProperty);
130. writeTableFile.WriteLine((tableList[i] **as** table).startTimeProperty);
131. writeTableFile.WriteLine((tableList[i] **as** table).endTimeProperty);
132. writeTableFile.WriteLine((tableList[i] **as** table).playerOneProperty);
133. writeTableFile.WriteLine((tableList[i] **as** table).playerTwoProperty);
134. }
135. writeTableFile.Close();
136. }
137. **public** **bool** assignNewTable(**double** playerOneUserID) //to assign new table if one player comes
138. {
139. ArrayList tableList = **new** ArrayList(); //ArrayList to store list of tables
140. tableList = readTableFileList(); //reads tables from file to list
142. **for** (**int** i = 0; i < tableList.Count; i++) //checks each table
143. {
144. **if** ((tableList[i] **as** table).gameStatus == 0) //if table is empty
145. {
146. (tableList[i] **as** table).gameStatusProperty = 1; //game status i.e. 0->Empty Table, 1->One Player Assigned, 2->Two Players Assigned
147. (tableList[i] **as** table).startTimeProperty = DateTime.Now;
148. (tableList[i] **as** table).endTimeProperty = DateTime.Now;
149. (tableList[i] **as** table).playerOneProperty = playerOneUserID;
150. (tableList[i] **as** table).playerTwoProperty = 0;
151. writeTableFileList(tableList); //write again to file
152. Console.WriteLine("Table Successfully Assigned.");
153. **return** **true**;
154. }
155. **else** **if** ((tableList[i] **as** table).gameStatus == 1) //if table has 1 playyer then assign the new player to this table
156. {
157. (tableList[i] **as** table).gameStatusProperty = 2; //game status i.e. 0->Empty Table, 1->One Player Assigned, 2->Two Players Assigned
158. (tableList[i] **as** table).startTimeProperty = DateTime.Now;
159. (tableList[i] **as** table).endTimeProperty = DateTime.Now;
160. (tableList[i] **as** table).playerTwoProperty = playerOneUserID;
161. writeTableFileList(tableList); //write again to file
162. Console.WriteLine("Table Successfully Assigned.");
163. **return** **true**;
164. }
165. }
166. **return** **false**;
167. }
168. **public** **bool** assignNewTable(**double** playerOneUserID, **double** playerTwoUserID) //to assign new table if two players come
169. {
170. ArrayList tableList = **new** ArrayList(); //ArrayList to store list of tables
171. tableList = readTableFileList(); //reads tables from file to list
173. **for** (**int** i = 0; i < tableList.Count; i++) //checks each table
174. {
175. **if** ((tableList[i] **as** table).gameStatus == 0) //if table is empty then assign to them
176. {
177. (tableList[i] **as** table).gameStatusProperty = 2; //game status i.e. 0->Empty Table, 1->One Player Assigned, 2->Two Players Assigned
178. (tableList[i] **as** table).startTimeProperty = DateTime.Now;
179. (tableList[i] **as** table).endTimeProperty = DateTime.Now;
180. (tableList[i] **as** table).playerOneProperty = playerOneUserID;
181. (tableList[i] **as** table).playerTwoProperty = playerTwoUserID;
182. writeTableFileList(tableList); //write again to file
183. Console.WriteLine("Table Successfully Assigned.");
184. **return** **true**;
185. }
186. }
187. **for** (**int** i = 0; i < tableList.Count; i++) //checks each table
188. {
189. **if** ((tableList[i] **as** table).gameStatus == 1) //as no full table is empty so now it will check table where one player is assigned so that they can start game immediately
190. {
191. (tableList[i] **as** table).gameStatusProperty = 2; //game status i.e. 0->Empty Table, 1->One Player Assigned, 2->Two Players Assigned
192. (tableList[i] **as** table).startTimeProperty = DateTime.Now;
193. (tableList[i] **as** table).endTimeProperty = DateTime.Now;
194. (tableList[i] **as** table).playerOneProperty = playerOneUserID;
195. (tableList[i] **as** table).playerTwoProperty = playerTwoUserID;
196. writeTableFileList(tableList);
197. Console.WriteLine("Table Successfully Assigned.");
198. **return** **true**;
199. }
200. }
201. **return** **false**;
202. }
203. **public** **void** displayTableList() //to display all tables status
204. {
205. ArrayList tableList = **new** ArrayList(); //ArrayList to store list of tables
206. tableList = readTableFileList(); //reads tables from file to list
208. Console.Write("--------------------------------------------------------------------------------");
209. Console.WriteLine("DISPLAY ALL TABLES STATUS:");
210. Console.Write("--------------------------------------------------------------------------------");
212. **for** (**int** i = 0; i < tableList.Count; i++) //checks each table
213. {
214. Console.WriteLine();
215. Console.WriteLine("----------------------");
216. Console.WriteLine("Table-ID: " + (tableList[i] **as** table).tableID);
217. Console.WriteLine("----------------------");
218. **if** ((tableList[i] **as** table).gameStatus == 0) //game status i.e. 0->Empty Table, 1->One Player Assigned, 2->Two Players Assigned
219. {
220. Console.WriteLine("0 Players Assigned.");
221. }
222. **else** **if** ((tableList[i] **as** table).gameStatus == 1) //game status i.e. 0->Empty Table, 1->One Player Assigned, 2->Two Players Assigned
223. {
224. player p = **new** player();
225. Console.WriteLine("1 Player Assigned.");
227. **string** playerOneName = p.fullName((tableList[i] **as** table).playerOneProperty);
228. Console.WriteLine("1. Player-1 (ID-" + (tableList[i] **as** table).playerOneProperty + ") " + playerOneName);
229. Console.WriteLine("Start Time:" + (tableList[i] **as** table).startTimeProperty);
230. }
231. **if** ((tableList[i] **as** table).gameStatus == 2) //game status i.e. 0->Empty Table, 1->One Player Assigned, 2->Two Players Assigned
232. {
233. player p = **new** player();
234. Console.WriteLine("2 Players Assigned.");
236. **string** playerOneName = p.fullName((tableList[i] **as** table).playerOneProperty);
237. **string** playerTwoName = p.fullName((tableList[i] **as** table).playerTwoProperty);
238. Console.WriteLine("1. Player-1 (ID-" + (tableList[i] **as** table).playerOneProperty + ") " + playerOneName);
239. Console.WriteLine("2. Player-2 (ID-" + (tableList[i] **as** table).playerTwoProperty + ") " + playerTwoName);
240. Console.WriteLine("Start Time:" + (tableList[i] **as** table).startTimeProperty);
241. }
242. }
243. }
244. **public** **void** submitTableResults() //to submit game results and clear table status
245. {
246. ArrayList tableList = **new** ArrayList(); //ArrayList to store list of tables
247. tableList = readTableFileList(); //reads tables from file to list
249. Console.Write("--------------------------------------------------------------------------------");
250. Console.WriteLine("SUBMIT TABLE RESULTS:");
251. Console.Write("--------------------------------------------------------------------------------");
253. Console.WriteLine("Enter Table-ID:");
254. **int** id = **int**.Parse(Console.ReadLine());
256. **for** (**int** i = 0; i < tableList.Count; i++) //checks each table
257. {
258. **if** ((tableList[i] **as** table).tableID == id) //checks for the required table
259. {
260. player p = **new** player();
261. Console.WriteLine("Select Won User:"); //asks for the won user
262. **string** playerOneName = p.fullName((tableList[i] **as** table).playerOneProperty);
263. **string** playerTwoName = p.fullName((tableList[i] **as** table).playerTwoProperty);
264. Console.WriteLine("1. Player-1 (ID-" + (tableList[i] **as** table).playerOneProperty + ") " + playerOneName);
265. Console.WriteLine("2. Player-2 (ID-" + (tableList[i] **as** table).playerTwoProperty + ") " + playerTwoName);
266. Console.WriteLine("3. Game Draw.");
267. **int** choice = **int**.Parse(Console.ReadLine());
268. **if** (choice == 1)
269. {
270. p.playerWon((tableList[i] **as** table).playerOneProperty); //updates won status of player1
271. p.playerLost((tableList[i] **as** table).playerTwoProperty); //updates lost status of player2
272. }
273. **else** **if** (choice == 2)
274. {
275. p.playerWon((tableList[i] **as** table).playerTwoProperty); //updates won status of player2
276. p.playerLost((tableList[i] **as** table).playerOneProperty); //updates lost status of player1
277. }
278. **else** **if** (choice == 3)
279. {
280. p.playerDraw((tableList[i] **as** table).playerOneProperty, (tableList[i] **as** table).playerTwoProperty); //updates draw status of both users
281. }
282. **else**
283. {
284. Console.WriteLine("ERROR!!! Invalid Input.");
285. }
286. (tableList[i] **as** table).endTimeProperty = DateTime.Now;
288. StreamWriter writeGameLogFile = **new** StreamWriter("GameLog.txt", **true**);
289. writeGameLogFile.WriteLine((tableList[i] **as** table).tableIDProperty);
290. writeGameLogFile.WriteLine((tableList[i] **as** table).playerOneProperty);
291. writeGameLogFile.WriteLine((tableList[i] **as** table).playerTwoProperty);
292. writeGameLogFile.WriteLine((tableList[i] **as** table).startTimeProperty);
293. writeGameLogFile.WriteLine((tableList[i] **as** table).endTimeProperty);
294. writeGameLogFile.Close();
296. (tableList[i] **as** table).gameStatus = 0; //clears table status to empty
297. (tableList[i] **as** table).playerOneProperty = 0; //clears table player1 to empty
298. (tableList[i] **as** table).playerTwoProperty = 0; //clears table player2 to empty
299. writeTableFileList(tableList);
300. **return**;
301. }
302. }
303. }
304. **public** **void** displayGameLogFile() //to display game history
305. {
306. Console.Write("--------------------------------------------------------------------------------");
307. Console.WriteLine("DISPLAY GAME LOG HISTORY:");
308. Console.Write("--------------------------------------------------------------------------------");
310. **double** id;
311. **double** player1ID;
312. **double** player2ID;
313. DateTime start;
314. DateTime end;
316. StreamReader readGameLogFile = **new** StreamReader("GameLog.txt");
317. **while** (!readGameLogFile.EndOfStream) //reads game log file till end
318. {
319. id = **double**.Parse(readGameLogFile.ReadLine());
320. player1ID = **double**.Parse(readGameLogFile.ReadLine());
321. player2ID = **double**.Parse(readGameLogFile.ReadLine());
322. start = DateTime.Parse(readGameLogFile.ReadLine());
323. end = DateTime.Parse(readGameLogFile.ReadLine());
325. Console.WriteLine("T-ID:" + id + " | P1-ID:" + player1ID + " | P2-ID:" + player2ID + " | START:" + start + " | END:" + end);
326. }
327. }
328. }
329. }

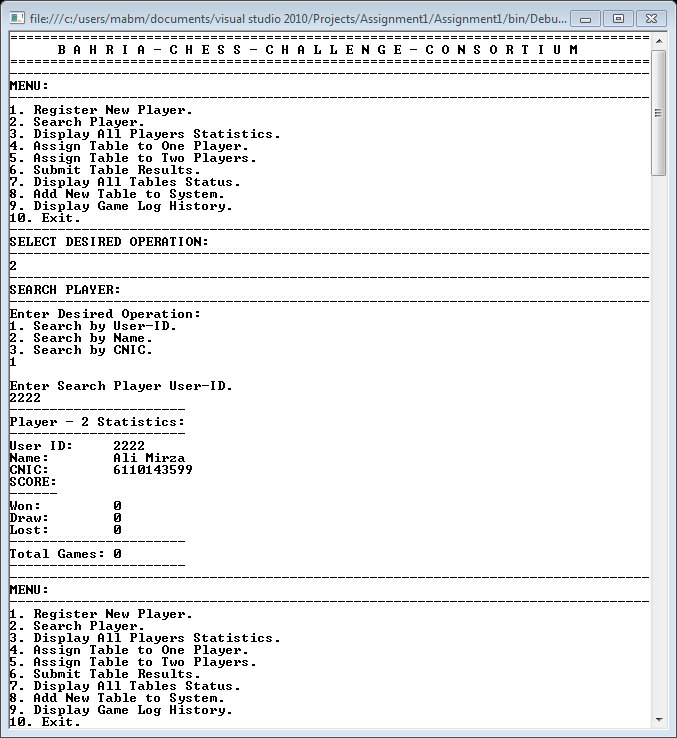
**Program.cs File:**

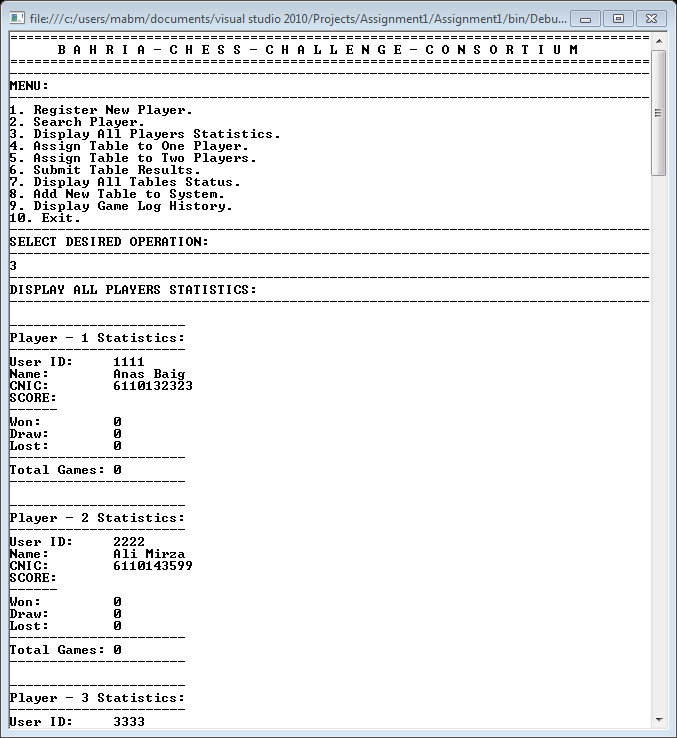
1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
6. **namespace** Assignment1
7. {
8. **class** Program
9. {
10. **static** **void** Main(**string**[] args)
11. {
12. Console.BackgroundColor = ConsoleColor.White;
13. Console.Clear();
14. Console.ForegroundColor = ConsoleColor.Black;
15. Console.Write("================================================================================");
16. Console.WriteLine("      B A H R I A - C H E S S - C H A L L E N G E - C O N S O R T I U M");
17. Console.Write("================================================================================");
19. consoleMenu c = **new** consoleMenu();
20. c.printConsoleMenu();
21. }
22. }
23. }

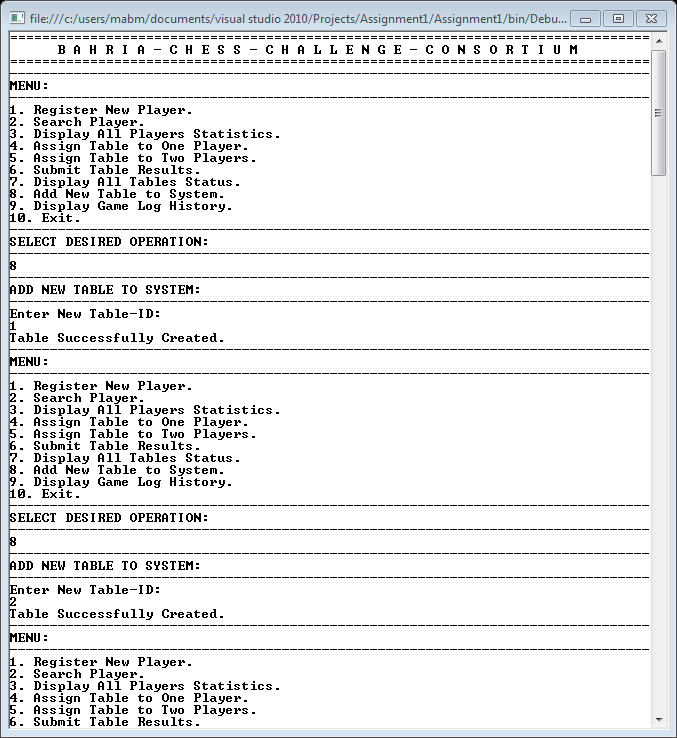
**Console Display Output:**

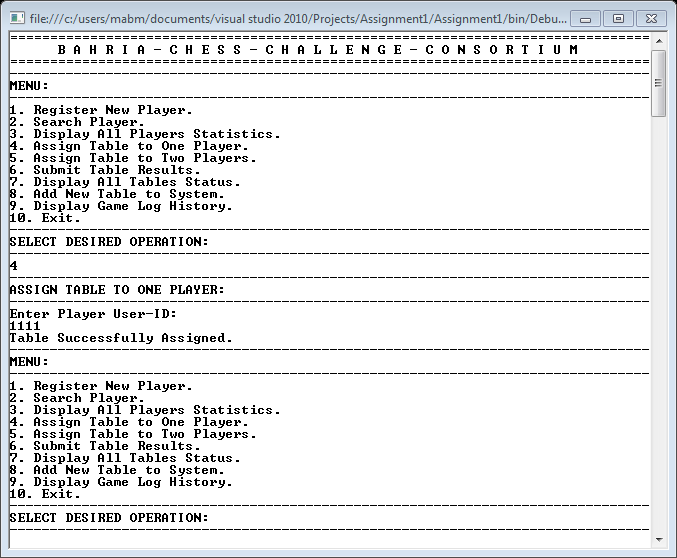
****

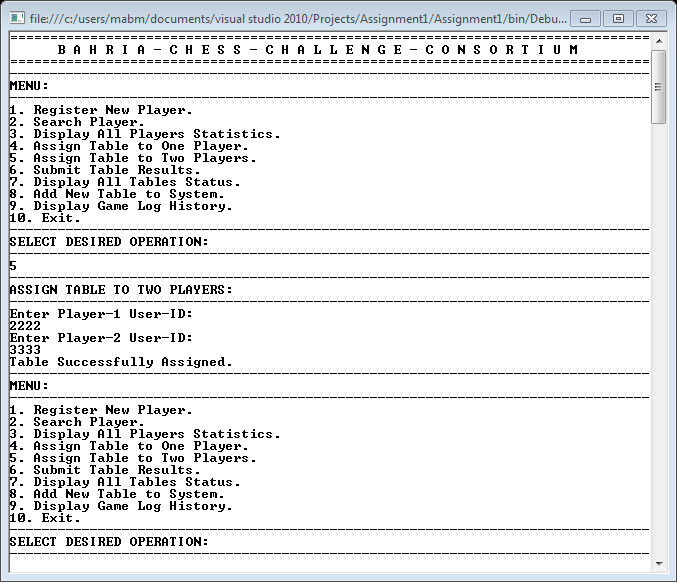
****

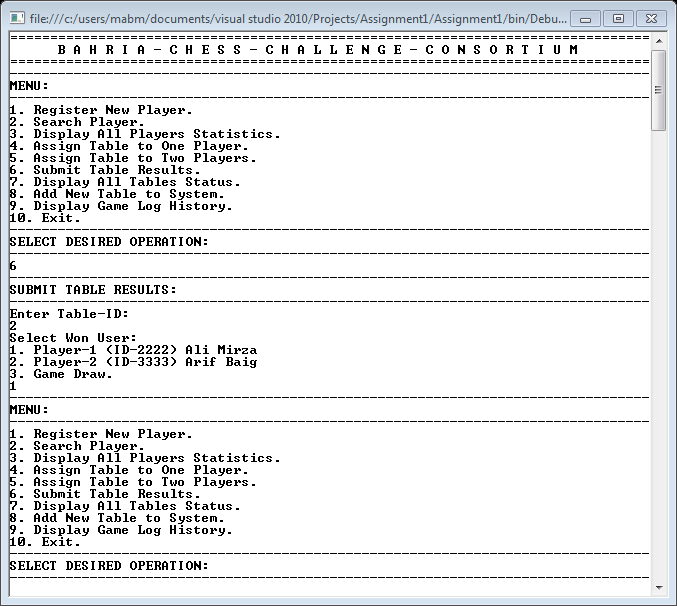
****

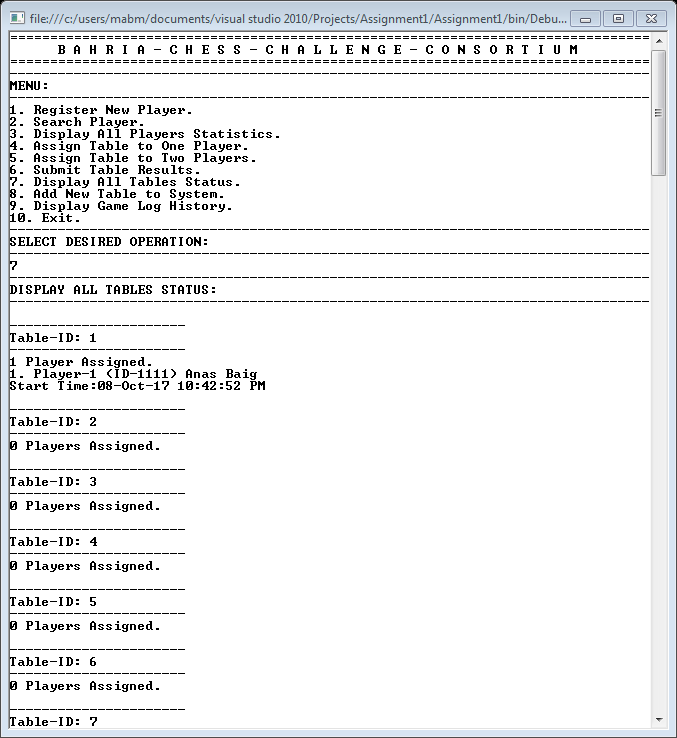
****

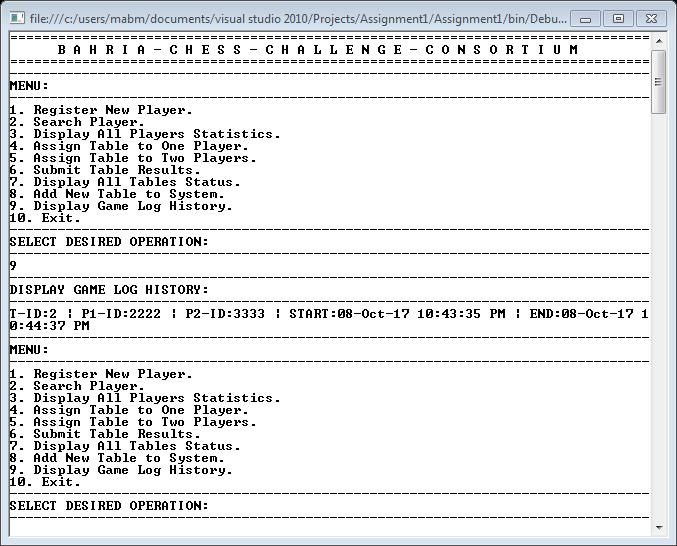
****

****

****

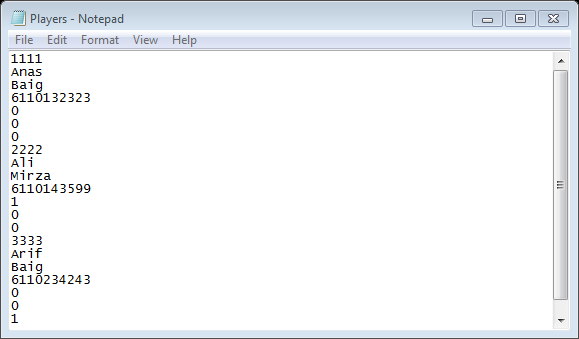
****

****

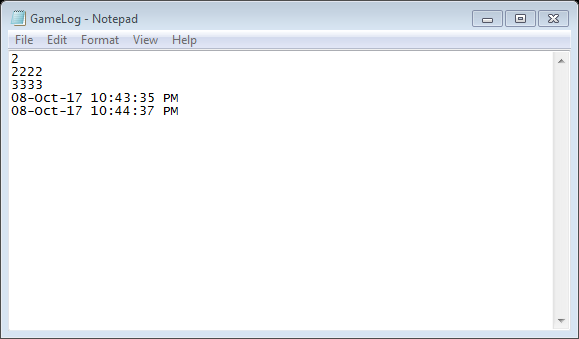
****

**Text File Output:**

**Players.txt File:**

****

**GameLog.txt File:**

****

**Tables.txt File:**

