Problem 1 User

Description

The player profile of a chess game contains name, login times, played matches, won matches, won ratio, rank(beginner, professional, expert). Implement a class named **User** with following operations: see the interfaces section. Its constructor should provide default player's rank and default name "guest". There may be multiple "guest" users at one time, never to mix up their login sessions.

Rank	Played matches(>=)	Won ratio(>=)
professional	50	70
expert	100	80

Interfaces

Static method:

- **User *getUser()** - To get the guest
- **User *getUser(string name)** - To get the user

Instance methods:

- **string getName()** - Get the player's name
- **void login()** - To login if the player is not online
- **void logout()** - To logout if the player is online
- **bool isOnline()** - Whether the user is online
- **int getLoginTimes()** - Get the login times
- **int getPlayedMatches()** - Get the count of played matches
- **int getWonMatches()** - Get the count of won matches
- **double getWonRatio()** - Get the player's won ratio, initial value should be 0
- **string getRank()** - Get the player's rank, the return should be one of ["beginner", "professional", "expert"]
- **void playWith(User *anotherUser, bool win)** - Play with other user, and win indicates the result. The current user will win the game if win parameter is true. **The result is valid only when the two users are both online.**

Notes

Your source code should not include the main function.

Sample test

```
User *guest0 = User::getUser();
User *guest1 = User::getUser();
guest0->login();
guest0->isOnline(); // true
guest0->logout();
guest0->isOnline(); // false
guest1->getLoginTimes(); // 0
```

```
User *maoge = User::getUser("Maoge");
maoge->getName(); // Maoge
maoge->login();
maoge->logout();
maoge->login();
maoge->login();
maoge->getLoginTimes(); // 2

User *maogeCopy = User::getUser("Maoge");
User *kuanye = User::getUser("Kuanye");
kuanye->playWith(maoge, true);
kuanye->getWonMatches(); // 0
kuanye->login();
kuanye->playWith(maoge, true);
kuanye->getWonMatches(); // 1

maogeCopy->getPlayedMatches(); // 1
```

Hint

static map to handle all instances

Source

Lecture 4 ppt

Problem 2 Matrix

Description

Implement the class Matrix of a 4X4 float matrix, providing interface as follows:

- matrix add
- matrix multiply
- print
- operator +=
- subscripting function object
  - float& operator()(int row, int column);
  - float operator()(int row, int column) const;
- Default constructor that accepts 16 values from standard device
- Another constructor that accept an array of size 16

Interfaces

Class name: **Matrix**

Instance methods:

- **void add(const Matrix&)**
- **void multiply(const Matrix&)**
- **void print()**

Operators:

- **Matrix& operator+=(const Matrix&)**
- **float& operator()(int row, int column)**
- **float operator()(int row, int column) const**

Notes

Your source code should not include the main function.

Sample test

```
float a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16};

Matrix m1(a);
Matrix m2(a);

m1.multiply(m2);
m1.print();
/*
90 100 110 120
202 228 254 280
314 356 398 440
426 484 542 600
*/
```

Source

Lecture 4 ppt

Problem 1 Player Profile

Description

The player profile of a chess game contains name, login times, played matches, won matches, won ratio, rank(beginner, professional, expert). Implement a class named **User** with following operations: see the interfaces section. Its constructor should provide default player's rank and default name "guest". There may be multiple "guest" users at one time, never to mix up their login sessions.

Rank	Played matches(>=)	Won ratio(>=)
professional	50	70
expert	100	80

Interfaces

Static method:

- **User *getUser()** - To get the guest
- **User *getUser(string name)** - To get the user

Instance methods:

- **string getName()** - Get the player's name
- **void login()** - To login if the player is not online
- **void logout()** - To logout if the player is online
- **bool isOnline()** - Whether the user is online
- **int getLoginTimes()** - Get the login times
- **int getPlayedMatches()** - Get the count of played matches
- **int getWonMatches()** - Get the count of won matches
- **double getWonRatio()** - Get the player's won ratio, initial value should be 0
- **string getRank()** - Get the player's rank, the return should be one of ["beginner", "professional", "expert"]
- **void playWith(User *anotherUser, bool win)** - Play with other user, and win indicates the result. The current user will win the game if win parameter is true. **The result is valid only when the two users are both online.**

Notes

Your source code should not include the main function.

Sample test

```
User *guest0 = User::getUser();
User *guest1 = User::getUser();
guest0->login();
guest0->isOnline(); // true
guest0->logout();
guest0->isOnline(); // false
guest1->getLoginTimes(); // 0

User *maoge = User::getUser("Maoge");
maoge->getName(); // Maoge
maoge->login();
maoge->logout();
maoge->login();
maoge->login();
maoge->getLoginTimes(); // 2

User *maogeCopy = User::getUser("Maoge");
User *kuanye = User::getUser("Kuanye");
kuanye->playWith(maoge, true);
kuanye->getWonMatches(); // 0
kuanye->login();
kuanye->playWith(maoge, true);
kuanye->getWonMatches(); // 1

maogeCopy->getPlayedMatches(); // 1
```

Hint

static map to handle all instances

Source

Lecture 4 ppt

Problem 2 Matrix

Description

Implement the class Matrix of a 4X4 float matrix, providing interface as follows:

- matrix add
- matrix multiply
- print
- operator +=
- subscripting function object
  - float& operator()(int row, int column);
  - float operator()(int row, int column) const;
- Default constructor that accepts 16 values from standard device
- Another constructor that accept an array of size 16

Interfaces

Class name: **Matrix**

Instance methods:

- **void add(const Matrix&)**
- **void multiply(const Matrix&)**
- **void print()**

Operators:

- **Matrix& operator+=(const Matrix&)**
- **float& operator()(int row, int column)**
- **float operator()(int row, int column) const**

Notes

Your source code should not include the main function.

Sample test

```
float a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16};

Matrix m1(a);
Matrix m2(a);

m1.multiply(m2);
m1.print();
/*
90 100 110 120
202 228 254 280
314 356 398 440
426 484 542 600
```

```
*/
```

Source

Lecture 4 ppt