Programming Exercise 09
Password Hashing and Authentication

C# Step by Step

You have been assigned the implementation of a password system that runs in memory. Write a console program that does two things: (1) it collects a plain text password from an individual user and stores the password in a string cipher, and (2) it verifies a user by username and password. Your user interface should have three options: (1) save a new password for a specific username, (2) authenticate a specific username/password pair, or (3) exit the application. On exit, all username/password pairs will be lost

Passwords entered at the console prompt are to be in plain text. All password saved in memory are to be encoded. Imagine that someone else is writing a database module to afford persistent storage of usernamne/password pairs. You do not want anyone with access to the database to be able to read the plain text of passwords.

You have three challenges in this exercise. First, you need to construct a user interface that presents the user with the three options listed above. Second, you need to discover how you can convert strings to unintelligible collections of random characters — this is called *hashing*. Check the documentation — it's your friend. Third, you need to define a data structure (collection) that can insert values and search values very rapidly. *In this programming exercise, you are required to use a* ***generic dictionary***. Make sure you add `using System.Collections.Generic` to your header.

As always, work the problem out by hand first, then implement your natural language solution in C# code. The exercise has some challenges, but it is decidedly on the easy side. Solve the problem first in English, then rewrite it in the language that the machine understands.

String Hashing: 70 points Write an application that allows a user to enter a string a console prompt, hashes the string (using, perhaps, MD5 or SHA1), and prints the plain text string and the hashed result to the console. You may use the following listing as a base application:

```
1  class Program
2  {
3      static void Main(string[] args)
4      {
5          int userResponse = 9;
6
7          while (userResponse != 0)
8          {
9              if (userResponse == 1)
10             {
11                 Util.getNewUser();
12                 userResponse = Util.printUI();
13             }
14             else if (userResponse == 2)
15             {
16                 Util.getUser();
17                 userResponse = Util.printUI();
18             }
19             else if (userResponse == 3)
20             {
21                 Util.printUsers();
22                 userResponse = Util.printUI();
23             }
24             else
25             {
26                 Console.WriteLine("Sorry, I didn't unerstand what you wanted to do.");
```

```
27                      userResponse = Util.printUI();
28                  }
29              };
30               System.Environment.Exit(0);
31          }
32  }
```

Interface Logic: 80 points Write a user interface that gives the user three options. Option 1 is to create an account by entering a username and a password. Option 2 is to authenticate a user by entering a username and password. Option 3 is to print all username/password pairs previously entered. Write `Util.printUI()`.

Perhaps the user interface might look something like this:

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

            PASSWORD AUTHENTICATION SYSTEM

            Please select one option:

            1. Establish an account
            2. Authenticate a user
            3. Exit the system

            Enter selection:

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

Collecting accounts: 90 points When a user establishes an account, the user will submit a unique usernamne and password. The application should perform three tasks. First, the application must determine whether the username is globally unique, that is, no duplicate usernames allowed. Second, the application must hash the password, that is, no passwords in plain text allowed. Third, the application must save the username/password pair in some kind of appropriate data structure. Users will then be returned to the initial interface.

Implement the following functions:

- printUI()
- getUser()
- getNewUser()
- printUsers()

Authenticating users: 100 points When a user seeks to authenticate, he enters a username/password pair. If the application cannot locate a user by the given username, it will indicate that the account does not exist. If the account exists, the application will compare the proffered password with the password paired with the username. If the passwords match, the application will print a message that the user has properly authenticated. Note that the *hashed* passwords must be compared, as the system has ho knowledge of plain passwords. If the passwords do not match, the application will print a message that the user has not authenticated. Users will then be returned to the initial interface.