COMSATS University Islamabad (CUI) Attock Campus

**C**

**Final Term**

**MUHAMMAD YAQOOB CIIT/FA21-BCS-044/ATK**

**Submitted To: Dr. Bilal Haider  
subject: CC**

**Question 6:**

**Code:**using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text.RegularExpressions;

namespace Question\_6

{

internal class Program

{

static void Main(string[] args)

{

// Part 1: User Input

Console.Write("Enter usernames (separated by commas): ");

string input = Console.ReadLine();

string[] usernames = input.Split(',');

// List to store results for saving to file

List<string> results = new List<string>();

int totalUsernames = usernames.Length;

int validUsernamesCount = 0;

int invalidUsernamesCount = 0;

List<string> invalidUsernames = new List<string>();

foreach (var username in usernames)

{

string usernameTrimmed = username.Trim();

bool isValid = ValidateUsername(usernameTrimmed, out string validationMessage, out string password);

string result = $"{usernameTrimmed} - {validationMessage}";

if (isValid)

{

validUsernamesCount++;

result += $"\nGenerated Password: {password} (Strength: {EvaluatePasswordStrength(password)})\n";

}

else

{

invalidUsernamesCount++;

invalidUsernames.Add(usernameTrimmed);

}

results.Add(result);

}

// Display results in console

Console.WriteLine("\nValidation Results:");

foreach (var result in results)

{

Console.WriteLine(result);

}

// Save results to file

SaveResultsToFile(results, totalUsernames, validUsernamesCount, invalidUsernamesCount);

// Retry for invalid usernames

if (invalidUsernamesCount > 0)

{

Console.Write("\nDo you want to retry invalid usernames? (y/n): ");

string retryResponse = Console.ReadLine().ToLower();

if (retryResponse == "y")

{

Console.Write("\nEnter invalid usernames: ");

string retryInput = Console.ReadLine();

string[] retryUsernames = retryInput.Split(',');

// Retry validation for invalid usernames

foreach (var retryUsername in retryUsernames)

{

string retryUsernameTrimmed = retryUsername.Trim();

bool isRetryValid = ValidateUsername(retryUsernameTrimmed, out string retryValidationMessage, out string retryPassword);

string retryResult = $"{retryUsernameTrimmed} - {retryValidationMessage}";

if (isRetryValid)

{

retryResult += $"\nGenerated Password: {retryPassword} (Strength: {EvaluatePasswordStrength(retryPassword)})\n";

}

results.Add(retryResult);

}

SaveResultsToFile(results, totalUsernames + retryUsernames.Length, validUsernamesCount, invalidUsernamesCount);

}

}

Console.WriteLine("Processing complete.");

}

// Username validation function

static bool ValidateUsername(string username, out string validationMessage, out string password)

{

password = string.Empty;

// Validation rules for username

if (username.Length < 5 || username.Length > 15)

{

validationMessage = "Username length must be between 5 and 15";

return false;

}

if (!Regex.IsMatch(username, @"^[A-Za-z][A-Za-z0-9\_]\*$"))

{

validationMessage = "Username must start with a letter and contain only letters, digits, and underscores";

return false;

}

// Count letters, digits, and underscores

int upperCaseCount = 0, lowerCaseCount = 0, digitCount = 0, underscoreCount = 0;

foreach (char c in username)

{

if (char.IsUpper(c)) upperCaseCount++;

if (char.IsLower(c)) lowerCaseCount++;

if (char.IsDigit(c)) digitCount++;

if (c == '\_') underscoreCount++;

}

validationMessage = $"Valid\nLetters: {upperCaseCount + lowerCaseCount} (Uppercase: {upperCaseCount}, Lowercase: {lowerCaseCount}), Digits: {digitCount}, Underscores: {underscoreCount}";

// Generate password

password = GeneratePassword();

return true;

}

// Password generation function

static string GeneratePassword()

{

Random rand = new Random();

List<char> password = new List<char>();

// Ensure at least 2 uppercase, 2 lowercase, 2 digits, 2 special characters

string upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

string lower = "abcdefghijklmnopqrstuvwxyz";

string digits = "0123456789";

string special = "!@#$%^&\*";

for (int i = 0; i < 2; i++)

{

password.Add(upper[rand.Next(upper.Length)]);

password.Add(lower[rand.Next(lower.Length)]);

password.Add(digits[rand.Next(digits.Length)]);

password.Add(special[rand.Next(special.Length)]);

}

// Fill the rest with any valid characters

string allChars = upper + lower + digits + special;

while (password.Count < 12)

{

password.Add(allChars[rand.Next(allChars.Length)]);

}

// Shuffle to ensure randomness

password.Sort((a, b) => rand.Next(-1, 2));

return new string(password.ToArray());

}

// Evaluate password strength

static string EvaluatePasswordStrength(string password)

{

int upperCount = 0, lowerCount = 0, digitCount = 0, specialCount = 0;

foreach (var c in password)

{

if (char.IsUpper(c)) upperCount++;

else if (char.IsLower(c)) lowerCount++;

else if (char.IsDigit(c)) digitCount++;

else if ("!@#$%^&\*".Contains(c)) specialCount++;

}

if (upperCount >= 2 && lowerCount >= 2 && digitCount >= 2 && specialCount >= 2 && password.Length == 12)

{

return "Strong";

}

if (password.Length >= 8)

{

return "Medium";

}

return "Weak";

}

// Save results to file

static void SaveResultsToFile(List<string> results, int totalUsernames, int validUsernamesCount, int invalidUsernamesCount)

{

string filePath = "UserDetails.txt";

using (StreamWriter sw = new StreamWriter(filePath))

{

sw.WriteLine("Validation Results:");

foreach (var result in results)

{

sw.WriteLine(result);

}

sw.WriteLine("\nSummary:");

sw.WriteLine($"- Total Usernames: {totalUsernames}");

sw.WriteLine($"- Valid Usernames: {validUsernamesCount}");

sw.WriteLine($"- Invalid Usernames: {invalidUsernamesCount}");

}

Console.WriteLine("\nResults saved to UserDetails.txt.");

}

}

}

Output:  
