****

**Lab Terminal Q:6**

**Submitted by:** Areeba Mehboob

**Reg #:** fa21-bcs-034

**Q6: Write a C# program using Regular Expressions (Regex) to perform the following tasks….**

**Code:**

using System;

using System.Collections.Generic;

using System.IO;

using System.Text.RegularExpressions;

class Program

{

// Function to validate the username

static bool ValidateUsername(string username, out string validationMessage)

{

// Regex to match username rules

Regex usernameRegex = new Regex(@"^[a-zA-Z][a-zA-Z0-9\_]{4,14}$");

if (!usernameRegex.IsMatch(username))

{

validationMessage = "Invalid Username";

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

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

else if (!char.IsLetter(username[0]))

validationMessage = "Username must start with a letter.";

else

validationMessage = "Username contains invalid characters. Only letters, digits, and underscores are allowed.";

return false;

}

validationMessage = "Valid Username";

return true;

}

// Function to generate a secure random password

static string GeneratePassword()

{

Random rand = new Random();

string upperChars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

string lowerChars = "abcdefghijklmnopqrstuvwxyz";

string digits = "0123456789";

string specialChars = "!@#$%^&\*()\_+-=[]{}|;:,.<>?";

// Ensure we have the required characters

string password = "";

password += upperChars[rand.Next(upperChars.Length)];

password += upperChars[rand.Next(upperChars.Length)];

password += lowerChars[rand.Next(lowerChars.Length)];

password += lowerChars[rand.Next(lowerChars.Length)];

password += digits[rand.Next(digits.Length)];

password += digits[rand.Next(digits.Length)];

password += specialChars[rand.Next(specialChars.Length)];

password += specialChars[rand.Next(specialChars.Length)];

// Fill the remaining characters randomly

string allChars = upperChars + lowerChars + digits + specialChars;

for (int i = password.Length; i < 12; i++)

{

password += allChars[rand.Next(allChars.Length)];

}

// Shuffle the password to make it more random

char[] passwordArray = password.ToCharArray();

for (int i = 0; i < passwordArray.Length; i++)

{

int j = rand.Next(i, passwordArray.Length);

char temp = passwordArray[i];

passwordArray[i] = passwordArray[j];

passwordArray[j] = temp;

}

return new string(passwordArray);

}

// Function to evaluate password strength

static string EvaluatePasswordStrength(string password)

{

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

foreach (char c in password)

{

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

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

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

else specialCount++;

}

if (upperCount >= 2 && lowerCount >= 2 && digitCount >= 2 && specialCount >= 2)

return "Strong";

else if (upperCount >= 1 && lowerCount >= 1 && digitCount >= 1 && specialCount >= 1)

return "Medium";

else

return "Weak";

}

// Function to save results to a file

static void SaveResultsToFile(List<string> results, string fileName)

{

File.WriteAllLines(fileName, results);

}

static void Main()

{

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

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

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

int totalUsernames = 0, validUsernames = 0, invalidUsernames = 0;

// Part 1: Input usernames and validation

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

string input = Console.ReadLine();

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

foreach (string username in usernameArray)

{

string trimmedUsername = username.Trim();

totalUsernames++;

string validationMessage;

if (ValidateUsername(trimmedUsername, out validationMessage))

{

validUsernames++;

validationResults.Add($"{trimmedUsername} - Valid");

// Count the number of uppercase letters, lowercase letters, digits, and underscores

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

foreach (char c in trimmedUsername)

{

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

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

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

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

}

passwordResults.Add($"Letters: {upperCount + lowerCount} (Uppercase: {upperCount}, Lowercase: {lowerCount}), Digits: {digitCount}, Underscores: {underscoreCount}");

// Generate password

string password = GeneratePassword();

string passwordStrength = EvaluatePasswordStrength(password);

passwordResults.Add($"Generated Password: {password} (Strength: {passwordStrength})");

}

else

{

invalidUsernames++;

validationResults.Add($"{trimmedUsername} - Invalid ({validationMessage})");

}

}

// Part 2: Display Results

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

foreach (string result in validationResults)

{

Console.WriteLine(result);

}

foreach (string result in passwordResults)

{

Console.WriteLine(result);

}

// Part 3: Save to file

string fileName = "UserDetails.txt";

List<string> fileResults = new List<string>

{

"Validation Results:"

};

fileResults.AddRange(validationResults);

fileResults.AddRange(passwordResults);

fileResults.Add($"Summary:");

fileResults.Add($"- Total Usernames: {totalUsernames}");

fileResults.Add($"- Valid Usernames: {validUsernames}");

fileResults.Add($"- Invalid Usernames: {invalidUsernames}");

SaveResultsToFile(fileResults, fileName);

Console.WriteLine($"\nResults saved to {fileName}");

// Part 4: Retry invalid usernames

if (invalidUsernames > 0)

{

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

string retryChoice = Console.ReadLine();

if (retryChoice.ToLower() == "y")

{

Console.WriteLine("Enter invalid usernames: ");

string retryInput = Console.ReadLine();

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

foreach (string retryUsername in retryUsernames)

{

string trimmedUsername = retryUsername.Trim();

string retryMessage;

if (ValidateUsername(trimmedUsername, out retryMessage))

{

validUsernames++;

validationResults.Add($"{trimmedUsername} - Valid");

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

foreach (char c in trimmedUsername)

{

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

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

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

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

}

passwordResults.Add($"Letters: {upperCount + lowerCount} (Uppercase: {upperCount}, Lowercase: {lowerCount}), Digits: {digitCount}, Underscores: {underscoreCount}");

string password = GeneratePassword();

string passwordStrength = EvaluatePasswordStrength(password);

passwordResults.Add($"Generated Password: {password} (Strength: {passwordStrength})");

}

else

{

invalidUsernames++;

validationResults.Add($"{trimmedUsername} - Invalid ({retryMessage})");

}

}

// Update the file with new results

fileResults.Clear();

fileResults.Add("Validation Results:");

fileResults.AddRange(validationResults);

fileResults.AddRange(passwordResults);

fileResults.Add($"Summary:");

fileResults.Add($"- Total Usernames: {totalUsernames + invalidUsernames}");

fileResults.Add($"- Valid Usernames: {validUsernames}");

fileResults.Add($"- Invalid Usernames: {invalidUsernames}");

SaveResultsToFile(fileResults, fileName);

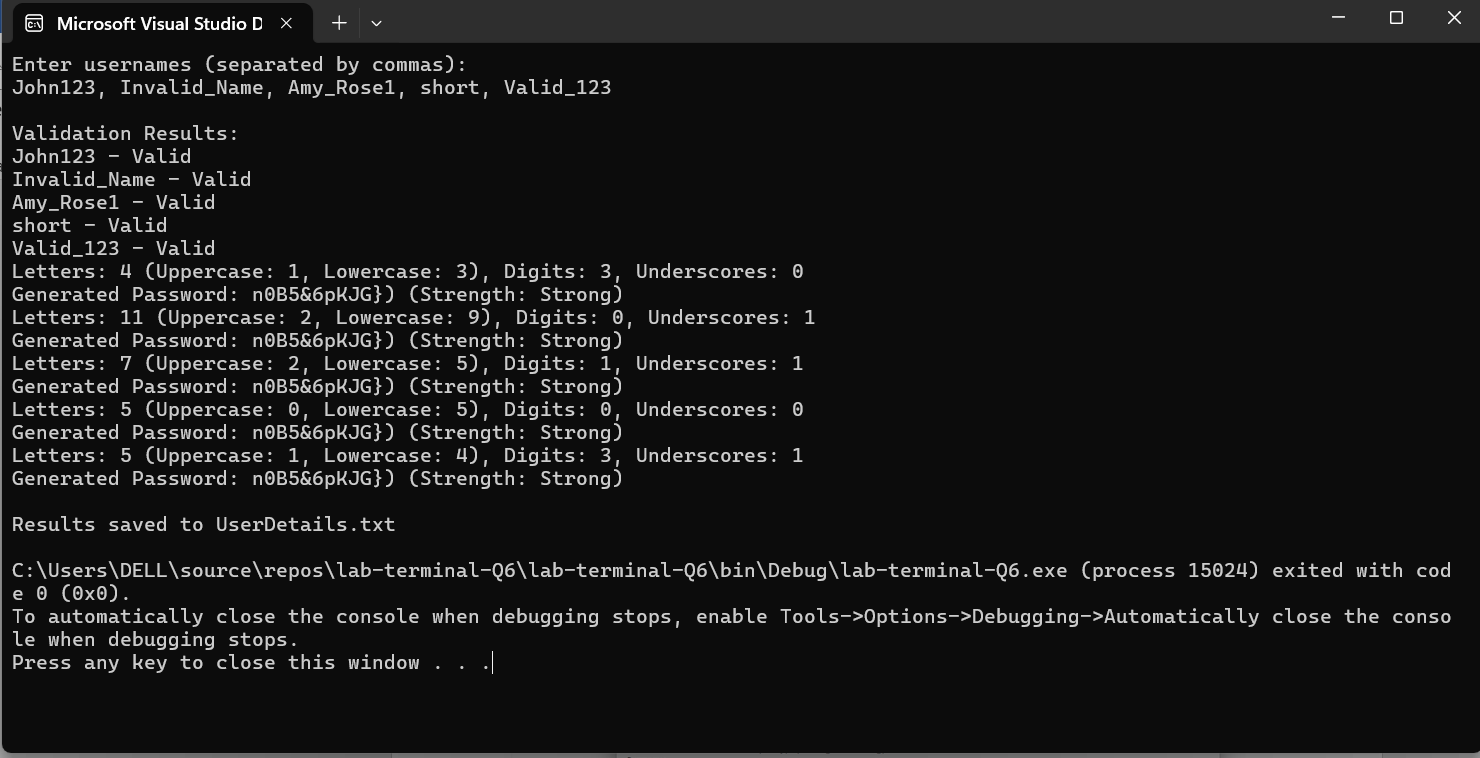
Console.WriteLine($"\nResults updated and saved to {fileName}");

}

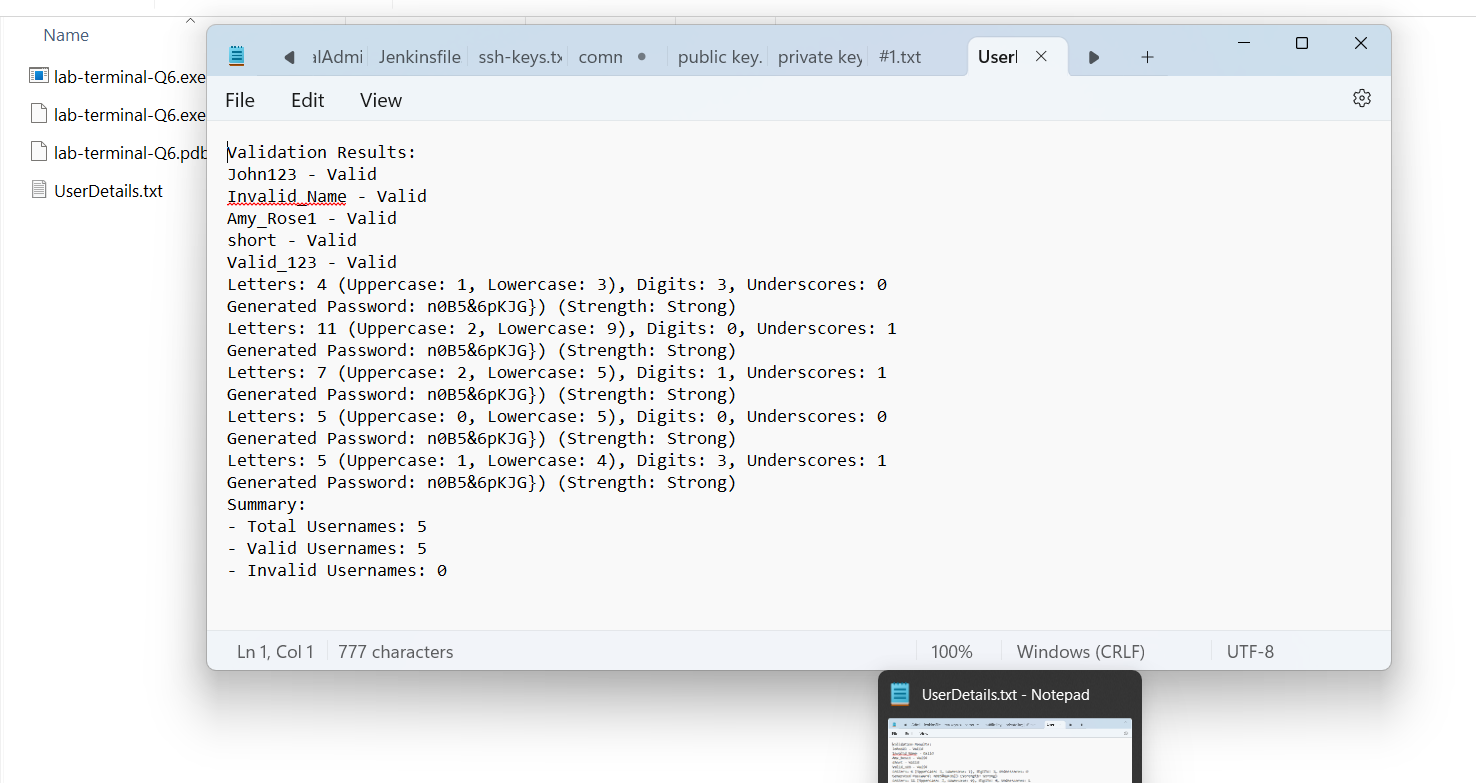
}

}

}

**output:  
**

**Results are saved to file: (UserDetails.txt)**

****