**Submitted By:** Shah Sarmad Hamdani

**Reg. no.:** fa21-bcs-021

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text.RegularExpressions;

class Program

{

    static void Main()

    {

        // Input user names

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

        string input = Console.ReadLine();

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

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

        List<(string Username, string Password, string Strength)> validUsernames = new List<(string, string, string)>();

        foreach (string username in usernames)

        {

            string trimmedUsername = username.Trim();

            var validationResult = ValidateUsername(trimmedUsername);

            if (validationResult.IsValid)

            {

                var password = GeneratePassword();

                string strength = CheckPasswordStrength(password);

                validUsernames.Add((trimmedUsername, password, strength));

                // Display valid username details

                Console.WriteLine($"{trimmedUsername} - Valid");

                Console.WriteLine($"Letters: {validationResult.LettersCount} (Uppercase: {validationResult.UppercaseCount}, Lowercase: {validationResult.LowercaseCount}), Digits: {validationResult.DigitCount}, Underscores: {validationResult.UnderscoreCount}");

                Console.WriteLine($"Generated Password: {password} (Strength: {strength})\n");

            }

            else

            {

                invalidUsernames.Add(trimmedUsername);

                Console.WriteLine($"{trimmedUsername} - Invalid ({validationResult.InvalidReason})\n");

            }

        }

        // Save to file

        SaveToFile(validUsernames, invalidUsernames);

        // Ask to retry invalid usernames

        if (invalidUsernames.Any())

        {

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

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

            if (retryChoice == "y")

            {

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

                string retryInput = Console.ReadLine();

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

                foreach (string retryUsername in retryUsernames)

                {

                    string trimmedUsername = retryUsername.Trim();

                    var validationResult = ValidateUsername(trimmedUsername);

                    if (validationResult.IsValid)

                    {

                        var password = GeneratePassword();

                        string strength = CheckPasswordStrength(password);

                        validUsernames.Add((trimmedUsername, password, strength));

                        Console.WriteLine($"{trimmedUsername} - Valid");

                        Console.WriteLine($"Generated Password: {password} (Strength: {strength})\n");

                    }

                    else

                    {

                        invalidUsernames.Add(trimmedUsername);

                        Console.WriteLine($"{trimmedUsername} - Invalid ({validationResult.InvalidReason})\n");

                    }

                }

                // Save results after retrying invalid usernames

                SaveToFile(validUsernames, invalidUsernames);

            }

        }

    }

    static (bool IsValid, int LettersCount, int UppercaseCount, int LowercaseCount, int DigitCount, int UnderscoreCount, string InvalidReason) ValidateUsername(string username)

    {

        // Regex to match username rules

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

        if (!regex.IsMatch(username))

        {

            if (!char.IsLetter(username[0])) return (false, 0, 0, 0, 0, 0, "Username must start with a letter");

            if (username.Length < 5 || username.Length > 15) return (false, 0, 0, 0, 0, 0, "Username length must be between 5 and 15 characters");

            return (false, 0, 0, 0, 0, 0, "Username contains invalid characters");

        }

        int uppercaseCount = username.Count(char.IsUpper);

        int lowercaseCount = username.Count(char.IsLower);

        int digitCount = username.Count(char.IsDigit);

        int underscoreCount = username.Count(c => c == '\_');

        return (true, username.Length, uppercaseCount, lowercaseCount, digitCount, underscoreCount, string.Empty);

    }

    static string GeneratePassword()

    {

        Random random = new Random();

        const string upperChars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

        const string lowerChars = "abcdefghijklmnopqrstuvwxyz";

        const string digits = "0123456789";

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

        const string allChars = upperChars + lowerChars + digits + specialChars;

        // Ensure password has at least 2 uppercase, 2 lowercase, 2 digits, and 2 special characters

        string password = $"{GetRandomCharacters(upperChars, 2)}{GetRandomCharacters(lowerChars, 2)}{GetRandomCharacters(digits, 2)}{GetRandomCharacters(specialChars, 2)}";

        password += GetRandomCharacters(allChars, 6); // Fill the rest with random characters

        return new string(password.OrderBy(c => random.Next()).ToArray()); // Shuffle the characters

    }

    static string GetRandomCharacters(string charSet, int length)

    {

        Random random = new Random();

        return new string(Enumerable.Range(0, length)

            .Select(\_ => charSet[random.Next(charSet.Length)])

            .ToArray());

    }

    static string CheckPasswordStrength(string password)

    {

        int lengthScore = password.Length >= 12 ? 3 : password.Length >= 8 ? 2 : 1;

        int varietyScore = password.Any(c => char.IsUpper(c)) && password.Any(c => char.IsLower(c)) &&

                           password.Any(c => char.IsDigit(c)) && password.Any(c => "!@#$%^&\*".Contains(c)) ? 3 : 2;

        if (lengthScore + varietyScore >= 5) return "Strong";

        if (lengthScore + varietyScore >= 4) return "Medium";

        return "Weak";

    }

    static void SaveToFile(List<(string Username, string Password, string Strength)> validUsernames, List<string> invalidUsernames)

    {

        using (StreamWriter writer = new StreamWriter("UserDetails.txt"))

        {

            writer.WriteLine("Validation Results:");

            foreach (var (username, password, strength) in validUsernames)

            {

                writer.WriteLine($"{username} - Valid");

                writer.WriteLine($"Generated Password: {password} (Strength: {strength})\n");

            }

            if (invalidUsernames.Any())

            {

                writer.WriteLine("Invalid Usernames:");

                foreach (var invalidUsername in invalidUsernames)

                {

                    writer.WriteLine(invalidUsername);

                }

            }

            writer.WriteLine("\nSummary:");

            writer.WriteLine($"Total Usernames: {validUsernames.Count + invalidUsernames.Count}");

            writer.WriteLine($"Valid Usernames: {validUsernames.Count}");

            writer.WriteLine($"Invalid Usernames: {invalidUsernames.Count}");

        }

        Console.WriteLine("Results saved to UserDetails.txt.");

    }

}