**COMPILER CONSTRUCTION**

**LAB TERMINAL**

**Student Name:** Hafza Nawaz

**Registration no:** FA21-BCS-018

**Teacher Name:** Sir Bilal Ahmad

**Question 6:**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Text.RegularExpressions;

class Program

{

static void Main()

{

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

string input = Console.ReadLine();

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

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

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

foreach (string username in usernames)

{

string tempUsername = username.Trim();

bool isValid = ValidateUsername(tempUsername, out string validationReason);

if (isValid)

{

validUsernames.Add(tempUsername);

string password = GeneratePassword(tempUsername);

string passwordStrength = CheckPasswordStrength(password);

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

Console.WriteLine($" Letters: {CountLetters(tempUsername)} (Uppercase: {CountUppercase(tempUsername)}, Lowercase: {CountLowercase(tempUsername)}), Digits: {CountDigits(tempUsername)}, Underscores: {CountUnderscores(tempUsername)}");

Console.WriteLine($" Generated Password: {password} (Strength: {passwordStrength})");

}

else

{

invalidUsernames.Add(tempUsername);

Console.WriteLine($"{tempUsername} - Invalid ({validationReason})");

}

}

Console.WriteLine("Summary:");

Console.WriteLine($"- Total Usernames: {usernames.Length}");

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

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

if (invalidUsernames.Count > 0)

{

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

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

if (retryChoice == "y")

{

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

string retryInput = Console.ReadLine();

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

foreach (string username in retryUsernames)

{

string tempUsername = username.Trim();

bool retryValid = ValidateUsername(tempUsername, out string retryValidationReason);

if (retryValid)

{

validUsernames.Add(tempUsername);

string retryPassword = GeneratePassword(tempUsername);

string retryPasswordStrength = CheckPasswordStrength(retryPassword);

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

Console.WriteLine($" Letters: {CountLetters(tempUsername)} (Uppercase: {CountUppercase(tempUsername)}, Lowercase: {CountLowercase(tempUsername)}), Digits: {CountDigits(tempUsername)}, Underscores: {CountUnderscores(tempUsername)}");

Console.WriteLine($" Generated Password: {retryPassword} (Strength: {retryPasswordStrength})");

}

else

{

invalidUsernames.Add(tempUsername);

Console.WriteLine($"{tempUsername} - Invalid ({retryValidationReason})");

}

}

}

}

SaveResultsToFile(validUsernames, invalidUsernames);

}

static bool ValidateUsername(string username, out string validationReason)

{

validationReason = string.Empty;

if (string.IsNullOrWhiteSpace(username))

{

validationReason = "Username cannot be empty.";

return false;

}

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

{

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

return false;

}

if (!Regex.IsMatch(username, @"^[a-zA-Z0-9\_]+$"))

{

validationReason = "Username can only contain letters, numbers, and underscores (\_).";

return false;

}

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

{

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

return false;

}

return true;

}

static string GeneratePassword(string username)

{

const string chars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&\*";

Random random = new Random();

StringBuilder password = new StringBuilder(12);

// Ensure at least 2 of each category

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

{

password.Append(chars[random.Next(26)]); // lowercase

password.Append(chars[random.Next(26, 52)]); // uppercase

password.Append(chars[random.Next(52, 62)]); // digits

password.Append(chars[random.Next(62, chars.Length)]); // special characters

}

// Fill the rest with random characters

while (password.Length < 12)

{

password.Append(chars[random.Next(chars.Length)]);

}

return password.ToString();

}

static string CheckPasswordStrength(string password)

{

int strength = 0;

if (password.Length >= 12) strength++;

if (password.Any(char.IsUpper)) strength++;

if (password.Any(char.IsLower)) strength++;

if (password.Any(char.IsDigit)) strength++;

if (password.Any(c => "!@#$%^&\*".Contains(c))) strength++;

return strength switch

{

1 => "Weak",

2 => "Medium",

3 => "Strong",

4 => "Very Strong",

\_ => "good",

};

}

static int CountLetters(string username) => username.Count(char.IsLetter);

static int CountUppercase(string username) => username.Count(char.IsUpper);

static int CountLowercase(string username) => username.Count(char.IsLower);

static int CountDigits(string username) => username.Count(char.IsDigit);

static int CountUnderscores(string username) => username.Count(c => c == '\_');

static void SaveResultsToFile(List<string> validUsernames, List<string> invalidUsernames)

{

// Use the provided desktop path to create the output file path

string filePath = Path.Combine("C:\\Users\\ABC\\Desktop", "UserDetails.txt");

using (StreamWriter writer = new StreamWriter(filePath))

{

writer.WriteLine("Validation Results:");

int index = 1;

foreach (string username in validUsernames)

{

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

writer.WriteLine($" Letters: {CountLetters(username)} (Uppercase: {CountUppercase(username)}, Lowercase: {CountLowercase(username)}), Digits: {CountDigits(username)}, Underscores: {CountUnderscores(username)}");

writer.WriteLine($" Generated Password: {GeneratePassword(username)} (Strength: {CheckPasswordStrength(GeneratePassword(username))})");

index++;

}

writer.WriteLine("Summary:");

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

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

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

}

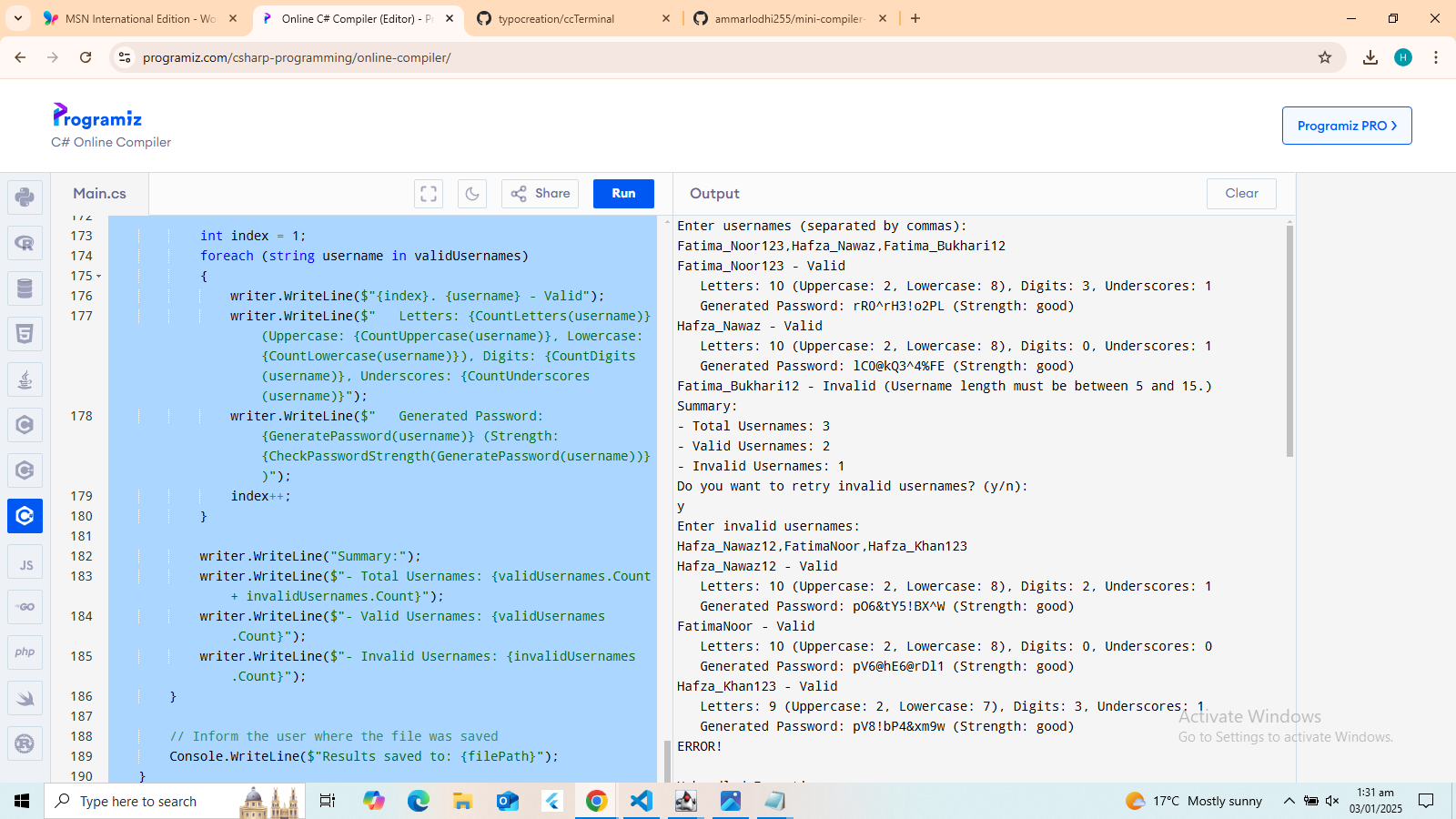
// Inform the user where the file was saved

Console.WriteLine($"Results saved to: {filePath}");

}

}

**Output:**

****