OBJECT ORIENTED PROGRAMMING

LAB ASSIGNMENT -2

Problem 1:

In today's digital world, secure passwords are essential to protect sensitive data. Weak passwords (like only numbers or only alphabets) are highly vulnerable to cyberattacks. A password checker is required to verify the **strength of a password** based on its composition.

The program should read a password from the user and analyze its characters using a **for loop**. It must count uppercase letters, lowercase letters, digits, and special characters. Based on these counts, the program will determine if the password is **Weak, Moderate, or Strong**.

Objectives:

- To use a **for loop** to traverse each character of the password string.
- To check and classify characters into:
    - Uppercase letters (A–Z)
    - Lowercase letters (a–z)
    - Digits (0–9)
    - Special characters (@, #, $, %, !, etc.)
- To decide password strength based on composition rules.
- To demonstrate practical use of loops in real-world **cybersecurity applications**.

Password Strength Rules:

- Weak:
    - Length < 6 OR contains only one type of character (e.g., only digits).
- Moderate:
    - Length ≥ 6 AND contains at least two types of characters (e.g., letters + digits).
- Strong:
    - Length ≥ 8 AND contains all four types (uppercase, lowercase, digits, special).

Expected Outcome:

- Program should count and display character categories.
- Based on rules, display the password's strength.
- Should handle different test cases (weak, moderate, strong).

Sample Input/Output

Input:

```
Enter password: abcd123
```

Output:

```
Uppercase Letters: 0
Lowercase Letters: 4
Digits: 3
Special Characters: 0
Password Strength: Moderate
```

Problem 2:

In real-world e-commerce or retail billing systems, customers purchase multiple items in a single transaction. The system must accept item prices one by one, calculate the total bill, and apply discounts if applicable. To simulate this scenario, we design a C++ program that repeatedly accepts item prices until the customer finishes shopping.

The loop should continue asking for prices until the user enters 0, which indicates checkout. Using break and continue statements ensures flexible control of the loop, similar to real-world billing systems.

Objectives:

- To simulate a shopping cart system where multiple item prices are entered.
- To use a for/while loop to repeatedly accept item prices.
- To apply:
  - continue → skip invalid entries (like negative prices).
  - break → stop when user enters 0 (checkout).
- To calculate the total bill and apply discounts if total exceeds a threshold.

Discount Rule:

- If total bill > ₹5000, apply 10% discount.
- Otherwise, no discount.

Expected Outcome:

- Program should allow the user to keep adding item prices.
- Negative prices should be ignored with a warning.
- Entering 0 should stop input and calculate the final bill.
- If applicable, discount should be applied automatically.

Sample Input/Output

Input/Run:

```
Enter item price (0 to checkout): 1200
Enter item price (0 to checkout): 1500
Enter item price (0 to checkout): -50
Invalid price! Skipped.
Enter item price (0 to checkout): 3000
Enter item price (0 to checkout): 0
```

Output:

```
Total Bill (before discount): ₹5700
Discount Applied: ₹570
Final Bill: ₹5130
```

Problem 3:

In real-world traffic management, signals are crucial to ensure road safety and regulate the flow of vehicles. Drivers must respond appropriately to the traffic light color. The problem is to design a program in C++ that simulates a traffic signal and provides instructions to the driver based on the signal color entered.

Objective:

- To develop a C++ program that takes the traffic signal color (Red, Yellow, or Green) as input.
- To implement decision control statements (if, else-if, else) for decision-making.
- To display the corresponding driving instruction:
  - Red → Stop
  - Yellow → Get Ready
  - Green → Go
- To handle invalid input using proper validation.

Expected Outcome:

- The program should correctly identify the input signal color and display the respective driving instruction.
- If the input is invalid (not Red, Yellow, or Green), the program should display an error message.
- The output should simulate a real-world traffic light system by guiding the driver's action.

Sample Input/Output

Input:

```
   Enter traffic light color: Red
```

Output:

```
Stop!
```

Input:

```
Enter traffic light color: Green
```

Output:

```
Go!
```

Input:

```
Enter traffic light color: Blue
```

Output:

```
Invalid input
```