National University of computer and emerging sciences

Object Oriented Programming (CL-1004)

Semester: Spring 2025
Section: BCS-2F
Course Instructor: Mr. Shafique Rehman

LAB # 08

Introduction to Friend Classes, Friend Functions, & Operator Overloading

Submission Guidelines for LAB # 08

Please complete all tasks sincerely and upload the **.cpp files for Questions 01 and 02**. **For Question 03, a Word file will be required,** where you must write your reasoning/justification along with the sample code and a screenshot of the sample output.

No submissions will be accepted after the deadline!

TASK # 01

In a modern smart home, different devices such as lights, fans, air conditioners, and security systems are controlled through a central automation hub. The system must manage various electronic appliances, allowing authorized users to monitor and control their power consumption, adjust settings, and schedule automated tasks.

However, sensitive security system controls should not be accessible to regular users and must only be accessed by authorized maintenance staff via a special diagnostic tool. The system should also be capable of calculating the power consumption of each appliance based on usage duration and device type.

Attributes & Functionality Requirements

1. Device Management

Each device should have:

- **Device ID** (string) – A unique identifier for each appliance.
- **Device Type** (string) – Specifies if it is a **light, fan, AC, or security system**.
- **Power Rating** (double) – Power consumption per hour in watts.
- **Status** (bool) – Whether the device is **ON or OFF**.
- A function **void turnOn()** to turn the device ON.
- A function **void turnOff()** to turn the device OFF.
- A function **double calculatePowerUsage(int hours)** to compute energy consumption:
  - **Lights** consume energy based on simple multiplication: powerRating * hours.
  - **Fans** have **variable speed settings** affecting power consumption (powerRating * hours * speedFactor).
  - **Air Conditioners** adjust power dynamically based on room temperature (powerRating * hours * (1 - (currentTemp - desiredTemp) / 100)).
  - **Security Systems** have **fixed power consumption**, but should include **a security status check** in power calculation.

2. User Management

Each user should have:

- **User ID** (string) – A unique identifier for each person.
- **User Role** (string) – Defines if the user is a **Regular User** or **Maintenance Staff**.
- **Access Level** (int) – Determines whether they can view/edit security devices.
- A function **void viewAccessibleDevices()** – Displays only the devices they are allowed to control.

3. Special Security Controls

- Security system details (such as **camera status and intrusion alerts**) should **not be accessible to regular users**.
- Only **maintenance staff** should have access through a **friend function**:
  - **void accessSecurityLogs()** – Allows **authorized personnel** to check security status.

Function Prototypes and Overriding Requirements

1. **void turnOn() and void turnOff()** should be present in all devices but should:
   - **Lights & Fans:** Simple ON/OFF state change.
   - **ACs:** Also adjust the **temperature settings**.
   - **Security Systems:** Require **authorization check** before turning OFF.
2. **double calculatePowerUsage(int hours)** should be **overridden**:
   - **Lights:** powerRating * hours.
   - **Fans:** powerRating * hours * speedFactor.
   - **ACs:** powerRating * hours * (1 - (currentTemp - desiredTemp) / 100).
   - **Security Systems: Fixed consumption + log security status**.
3. **Friend Function: void accessSecurityLogs()**
   - Allows **only maintenance staff** to **view logs and system diagnostics**.
   - Regular users **cannot see logs** even if they try.
4. **Friend Class: MaintenanceTool**
   - **Friend of Security Systems**, allowing **maintenance personnel** to reset security alerts.

Sample Input & Expected Output

Input:

```
// User Setup
User user1("U001", "Regular User", 1);
User user2("U002", "Maintenance Staff", 2);

// Devices Setup
Light light1("L001", "LED Light", 10);
Fan fan1("F001", "Ceiling Fan", 75, 3);
AirConditioner ac1("AC001", "Split AC", 1500, 25);
SecuritySystem secSys1("S001", "Home Alarm", 100);

// Device Operations
```

```
light1.turnOn();
fan1.turnOn();
ac1.turnOn();
secSys1.turnOn();

// Power Consumption Calculation
cout << light1.calculatePowerUsage(5) << endl;  // 50 Watts
cout << fan1.calculatePowerUsage(3) << endl;    // Based on speed factor
cout << ac1.calculatePowerUsage(6) << endl;     // Adjusted for temp
cout << secSys1.calculatePowerUsage(24) << endl; // Fixed consumption

// Attempting to access security logs
user1.viewAccessibleDevices();  // Can see lights, fans, ACs, but NOT security systems
user2.accessSecurityLogs();       // Allowed to view security logs
```

Expected Output:

```
LED Light [ID: L001] turned ON.
Ceiling Fan [ID: F001] turned ON at Speed 3.
Split AC [ID: AC001] turned ON. Cooling to 25°C.
Home Alarm [ID: S001] activated.

Power Consumption:
LED Light: 50 Watts
Ceiling Fan: 225 Watts
Split AC: 7200 Watts
Security System: 2400 Watts (Includes security checks)

User: U001 - Accessible Devices:
- LED Light
- Ceiling Fan
- Split AC

User: U002 - Security Logs Accessed:
- Intrusion Detected at 3:00 AM
- System Reset Required
```

TASK # 02

A **Banking System** manages accounts with different types of transactions. Each account has a **balance (double)** and can perform deposits and withdrawals. To simplify operations, the system allows:

1. **Adding balances of two accounts** (+ operator).
2. **Transferring an amount from one account to another** (-= operator).
3. **Comparing two accounts** to check which has a higher balance (> operator).
4. **Displaying account details using stream insertion (<< operator)**.

Attributes and Function Prototypes:

Attributes:

- **accountNumber** (string) → Unique identifier for the account.
- **accountHolder** (string) → Name of the account owner.
- **balance** (double) → Current account balance.

Function Prototypes and Their Purpose:

```
Account operator+(const Account &); // Adds balances of two accounts.
Account &operator-=(double amount); // Transfers amount from one account.
bool operator>(const Account &); // Compares two accounts' balances.
friend ostream &operator<<(ostream &, const Account &); // Displays account details.
```

Sample Input & Expected Output:

Sample Input:

Account 1: Shafique Rehman, Balance: 5000
Account 2: Talha , Balance: 3000

Performing:
1. Adding Account 1 and Account 2 balances.
2. Transferring 2000 from Account 1 to Account 2.
3. Checking which account has a higher balance.
4. Displaying final account details.

Expected Output:

Total Balance (After Addition): 8000

After Transfer:
Shafique Reh's New Balance: 3000
Talha's New Balance: 5000

Comparison:
Shafique Rehman has less balance than Talha.

Final Account Details:
Account Number: 101 | Holder: Shafique Rehma | Balance: $3000
Account Number: 102 | Holder: Talha | Balance: $5000

TASK # 03

Question 1: Can a friend function be used to overload an operator that modifies the invoking object?

Problem Statement:
Consider the += operator, which modifies the left-hand operand. **Can a friend function be used to overload this operator?**

- If yes, how should it be implemented?
- If no, what alternative approach should be used?

Justify your answer with supporting C++ code.

Question 2: Is it possible to overload an operator using a friend function if one of the operands is a primitive data type?

Problem Statement:
Suppose we want to overload the + operator to allow addition between an object and a primitive type (e.g., object + int).

- **Can a friend function handle this case?**
- If yes, how would it be implemented?
- If no, what limitations exist?

Justify your answer with supporting C++ code.

Question 3: Can a friend function access private and protected members of a class without using an object of that class?

Problem Statement:
A friend function is granted access to private and protected members of a class.

- **Does it always need an object to access these members?**
- **Can a friend function access them directly without an object?**
- **Under what conditions might it fail?**

Justify your answer with supporting C++ code.

The End!