ಭಾರತೀಯ ಮಾಹಿತಿ ತಂತ್ರಜ್ಞಾನ ಸಂಸ್ಥೆ ರಾಯಚೂರು
भारतीय सूचना प्रौद्योगिकी संस्थान रायचूर
Indian Institute of Information Technology Raichur

LAB-6

Introduction to OOP

Submission Guidelines

- Ensure your system is in 'No Aeroplane Mode'.
- No Taskbar should be open.
- Create a new folder named `LAB-6`.
- Inside `LAB-6`, create two question files named in the following format: `[RollNumber]_[LabName]_[QuestionNumber]` (e.g., `12345_MatrixLab_Q1.cpp` and `12345_MatrixLab_Q2.cpp`)

Lab Timing and Submission

- Lab Time: **10:30AM to 12:50PM**
- Submission Deadline:1:00PM (Submit on Classroom)
- No Extensions: Late submissions will not be accepted.
- Viva: 1:05 PM

Question 1:-

You are tasked with building an **Inventory Management System** that manages different types of products using **Class Templates**, **Inheritance**, and **Polymorphism**. The system should be able to store different product types like `Electronics` and `Clothing`, each having some common properties and some unique ones.

🔧 Requirements

1. **Create a template abstract base class** `Product<T>` with:
    - ○ `T productID`
    - ○ `string name`
    - ○ `float price`
    - ○ Pure virtual function: `void displayDetails()`

2. **Create derived template classes**:

    - `Electronics<T>` with additional attributes:

        - `int warrantyInMonths`

        - `string brand`

    - `Clothing<T>` with additional attributes:

        - `string size` (S, M, L, XL)

        - `string fabricType`

3. Implement all constructors and override `displayDetails()` to display complete info for each type.

4. In the main function, maintain an **array of base class pointers** (using `Product<T>*`) and store mixed items (electronics and clothing). Dynamically create objects based on input.

📥 Input Format

- First line: Integer `n` (number of products)
- Next `n` blocks:
    - Type: `"Electronics"` or `"Clothing"`
    - Product ID (`int` for Electronics, `string` for Clothing)
    - Name
    - Price
    - Brand & Warranty (if Electronics)
    - Size & Fabric (if Clothing)

📤 Output Format

Display the details of each product using polymorphism:

[Product Type]
Product ID: <id>
Name: <name>
Price: $<price>
<other info>

Constraints

- `1 ≤ n ≤ 50`
- ProductID: int (100–9999) or string (3–10 characters)
- Name/Brand/Fabric: max 30 characters, no spaces
- Price: 1.00 – 10000.00
- Warranty: 0 – 60 months
- Size: One of [S, M, L, XL]

Sample Input:

```
3
Electronics 1010 Laptop 59999.99 HP 24
Clothing C123 TShirt 999.5 M Cotton
Electronics 1022 Smartphone 45999.49 Samsung 12
```

Sample Output:

```
[Electronics]
Product ID: 1010
Name: Laptop
Price: $59999.99
Brand: HP
Warranty: 24 months

[Clothing]
Product ID: C123
Name: TShirt
Price: $999.5
Size: M
Fabric: Cotton
```

[Electronics]

Product ID: 1022

Name: Smartphone

Price: $45999.49

Brand: Samsung

Warranty: 12 months

Question 2:-

You are tasked with designing and implementing software for a **Smart Sensor Network** used in **agriculture** to monitor environmental conditions. The system collects and manages data from various types of sensors, each of which has a unique ID and stores specific environmental readings.

The **types of sensors** include:

1. **TemperatureSensor -** Measures temperature (in degrees Celsius).

2. **HumiditySensor** - Measures humidity (in percentage).
3. **SoilMoistureSensor** - Measures soil moisture content (in percentage).

Each sensor has the following attributes:

- A **unique sensor ID**, which can be of any type (e.g., `int`, `string`, or `long`).
- A **location** (a string) representing the place where the sensor is deployed (e.g., `Field A`, `Greenhouse`, etc.).
- A **sensor reading**, which is sensor-specific (e.g., temperature value, humidity percentage, or soil moisture level).
- A **method to report data** in a structured and readable format.

Your task is to **design a flexible and extensible system** using **C++ Templates** and **Inheritance**, ensuring that it can handle multiple sensor types and **different ID types**. Additionally, the system should be capable of **polymorphic behavior** to handle diverse sensor objects uniformly.

Detailed Requirements

1. Base Class: Sensor<T>

- Create a **templated base class** `Sensor<T>` where:
  - `T` represents the type of sensor ID (e.g., `int`, `string`, `long`).
  - **Attributes**:

    - `sensorID`: A member of type `T` to store the unique identifier for the sensor.

    - `location`: A member of type `string` to represent the location of the sensor (e.g., `Field A`, `Greenhouse`).

  - **Methods**:

A **pure virtual function** `reportData()` that must be overridden in each derived class. This method should print a structured output, such as:

```
Sensor[ID: 101, Location: Field A] - Temperature: 27.5°C
```

2. Derived Classes for Each Sensor Type

- **TemperatureSensor**: Inherit from `Sensor<T>`, and add a member to store the temperature reading. Override `reportData()` to return a formatted string with the temperature value.

- **HumiditySensor**: Inherit from `Sensor<T>`, and add a member to store the humidity reading. Override `reportData()` to return a formatted string with the humidity percentage.

- **SoilMoistureSensor**: Inherit from `Sensor<T>`, and add a member to store the soil moisture reading. Override `reportData()` to return a formatted string with the soil moisture percentage.

3. Polymorphic Behavior

- In your `main()` function, create a **vector of `SensorBase*`** to store different sensor objects. These sensors will be of different types (Temperature, Humidity, SoilMoisture) and could have different ID types (e.g., `int`, `string`).

- Use **polymorphism** to call the `reportData()` function for each sensor object, regardless of the sensor type or ID type.

4. Sorting Sensors

- Implement a function to **sort the sensors by their ID** using templates. The sorting should work correctly for **different ID types** (e.g., integer IDs or string IDs). Implement this functionality in the `main()` function using the `std::sort()` algorithm.

- Ensure that your solution is flexible and can be easily extended to handle new types of sensors in the future.

Constraints

1. **ID Type Flexibility**: Your system should be able to handle different types for `sensorID` (e.g., `int`, `long`, `string`).

2. **Memory Management**: Use dynamic memory allocation (`new`/`delete`) for sensor objects to demonstrate good memory management practices. Avoid memory leaks.

3. **No Use of Global Variables**: Use only local variables and functions.

4. **Function Signatures**: Ensure the `reportData()` method has the correct signature in the base and derived classes.

Deliverables

- A C++ program that:
    1. Implements a templated base class `Sensor<T>`.
    2. Defines derived classes (`TemperatureSensor`, `HumiditySensor`, `SoilMoistureSensor`).
    3. Creates a collection of sensors with polymorphic behavior.
    4. Optionally sorts the sensors based on their ID.
    5. Uses proper memory management practices.

Input format :

4

Temperature 101 FieldA 27.5

Humidity HUM_01 FieldB 45.3

SoilMoisture 100 FieldC 32.1

Temperature TEMP_02 Greenhouse 22.0

Output format :

Assume you have a few sensors:

- `Sensor[ID: 101, Location: Field A] - Temperature: 27.5°C`

- `Sensor[ID: 102, Location: Field B] - Humidity: 45.3%`
- `Sensor[ID: 103, Location: Field C] - Soil Moisture: 32.1%`
- `Sensor[ID: 100, Location: Greenhouse] - Temperature: 22.0°C`

```
Sensor[ID: 100, Location: Greenhouse] - Temperature: 22.0°C
Sensor[ID: 101, Location: Field A] - Temperature: 27.5°C
Sensor[ID: 102, Location: Field B] - Humidity: 45.3%
Sensor[ID: 103, Location: Field C] - Soil Moisture: 32.1%
```