Lab 3: Defining Simple Classes

Course Code: CSE 4302
Week: 3
Topics: Class Definition, Access Specifier, and Accessing Members of an Object
Instructors:
Md. Bakhtiar Hasan, Assistant Professor, CSE
Ishmam Tashdeed, Lecturer, CSE
Submission Deadline: 12:05 P.M., November 26, 2025

Upload and Turn In your `.cpp` files to **Lab 3** on **Google Classroom**. If you are unable to finish all the tasks, **submit the tasks you have completed within the given time** (You have the chance to complete the remaining ones individually later on and demonstrate them during the next week's lab evaluation). No late submissions are allowed.
DO NOT forget to rename the files like `XXX_T1.cpp`, `XXX_T2.cpp`, ... (Replace `XXX` with the last three digits of your student ID).
After the submission deadline, you will undergo a one-on-one evaluation by one of the course instructors. The instructor will ask you to show your completed tasks on Google Classroom. They may ask you to run the code locally and ask you questions about it.

Lab Tasks

Institute of Unstable Time (IUT) operates a fleet of autonomous robots that restock, vend, record usage, and monitor perishable storage across campus. Your tasks today will design small classes to help with various functionalities carried out by those robots.
General Instructions:

- All data members must be private.
- If a setter receives invalid input, do not change state and print the exact error message given for that function.
- Add sensible defaults where noted.

1. Step It Up!

Create a class `BatchMeter` that models a simple counter used by a robot to count discrete supply batches. The instance of the class stores a *total* and a *pulse size* indicating how many units have been counted so far and how many units each pulse represents, respectively. You have to implement the operations to change the pulse size, register a pulse, read the total, and clear the counter.
The class should have the following functions:

- `void setPulseSize(int s)`: Sets the pulse size to `s`. `s` must be greater than `0`. Otherwise, it should not change the state and print: "`Invalid pulse; must be > 0`".

- `int getTotal()`: Returns the current total count.
- `void addPulse()`: Increase the total by the current pulse size.
- `void clear()`: Resets the total to `0`.

Notes: The default pulse size should be `1` and the default total should be `0`.

2. Location, Location, Location

Create a class `GridNavigator` that stores a simple 2D coordinate (x, y) representing a robot's current location on a campus map grid. The class should allow moving the robot and calculating the straight-line distance to another point. You have to implement movement, a reset function, and a distance calculator.

The class should have the following functions:

- `void setHome(double x, double y)`: Sets the "home" or initial coordinate. This point is stored separately and used for resetting.
- `void moveBy(double dx, double dy)`: Moves the robot from its current position by the offsets `dx` and `dy`.
- `void resetToHome()`: Resets the current coordinates to the home position.
- `double getX(), double getY()`: Return the respective current coordinate
- `double distanceTo(double x, double y)`: Returns the straight-line distance from the robot's current position to the target coordinate `(x, y)`. The formula for distance is

$$\sqrt{\left(x_2 - x_1\right)^2 + \left(y_2 - y_1\right)^2}$$

- `void printLocation()`: Prints the current location in the format: "`Position: (X.XX, Y.YY)`"

Notes: The default home position is `(0.0, 0.0)`. The robot starts at its home position.

3. Stock and Signal

Create a class `SupplyAlert` that monitors the stock level of a specific item in a vending machine or storage unit. It stores a current stock, a low stock threshold, and an item ID. The class should trigger an alert when the stock falls at or below the threshold. You have to implement functions to adjust the stock, set the threshold, and check the alert status.

The class should have the following functions:

- `void setThreshold(int t)`: Sets the low stock threshold to `t`. `t` must be greater than `0`. Otherwise, print "`Invalid threshold; must be > 0`" and do not change the state.
- `void setItemID(const char id[])`: Initializes the item identifier. Use a `char` array of size 20 or `std::string`.
- `void addStock(int n)`: Increases the current stock by `n`. `n` must be positive; otherwise, print "`Invalid stock addition`" and do nothing.
- `void consume(int n)`: Decreases the current stock by `n`. If `n` is negative or would make the stock negative, print "`Invalid consumption`" and do not change the stock.

- `bool isLow()`: Returns true if the current stock is at or below the low stock threshold, otherwise false.
- `void profile()`: Print a one-line summary: `"Item [ID]: Stock [current], Threshold [threshold], [OK/LOW]".` Print `"LOW"` if `isLow()` is true and `"OK"` otherwise.

Notes: Default threshold is 5. Default stock is 0.

4. Snack O'Clock

Create a class EventTime that models a robot's clock (hour, minute, second). The class should accept large or out-of-range inputs and normalize them so the hour is in-between 0 to 23, the minute is in-between 0 to 59, and the second is in-between 0 to 59. Remember to carry over from seconds to minutes, minutes to hours; hours wrap modulo 24. You have to implement getters, a robust setter, an advance method, and a formatted print.

The class should have the following functions:

- `int getHour()`, `int getMinute()`, `int getSecond()`: Return respective components.
- `void setTime(long h, long m, long s)`: Set time from `h`, `m`, `s`, which may be arbitrarily large and non-negative. Normalize so stored time is valid: seconds and minutes reduced to 0-59 with carries; hours reduced module 24. Example: `s = 125` should be `2` minutes and `5` seconds.
- `void addDuration(long h, long m, long s)`: Advance current time by given durations. Use the same normalization logic, with a 24-hour wrap for hours.
- `void printTime()`: Print time as `HH:MM:SS` (two digits each; leading zeros, if needed).

Notes: Accept only non-negative inputs. Do not use negative durations. Behavior is undefined for negatives unless you choose to handle them explicitly.