

# COS30008: Data Structures and Patterns

## Problem Set 1

Weighting: This assessment contributes 5% to the overall unit marks.

Learning Outcome(s) assessed: 1.

You can create task 1 folder, task 2 folder and etc., and zip all folders in one file (PS1\_<StudentID>). All the folders can only contain cpp and h files.

Use comments in your source files to explain what your algorithm is doing, step-by-step.

Create a brief **Report** to identify which Tasks you have completed and describe how your code works accompanied by screenshots of the Console output as evidence.

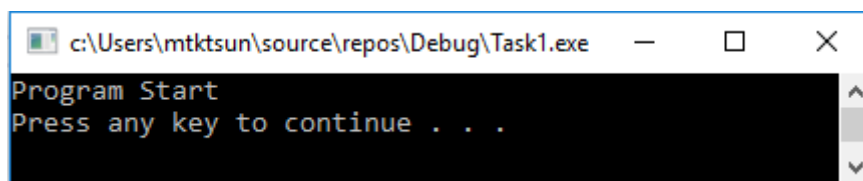
## C++ Aptitude and OOP Exercise

In Tower Defense, MOBAs, and RTS game design, in-game entities are often referred to as Units. A Unit can belong to one of the Human or Computer players. A Unit can be spawned as one of the variety of Unit Types available in a game (e.g. in StarCraft: Wraith, SCV, Ghost, Dragoon, Protoss Carrier, Hydralisk, etc.). Each Unit has its own internal data structure to contain its in-game Position, Health (Hitpoints), Energy, Shields, Movement Speed, Attack Speed, Attack Damage, Owner, Game State and more attributes.

For more background in Strategy Game Design: [https://en.wikipedia.org/wiki/Strategy\\_video\\_game](https://en.wikipedia.org/wiki/Strategy_video_game)

### Task 1 (2 mark)

Make a new empty project called ProblemSet1. Add a new source file named "Main.cpp". Create a Main loop in this source file, and include the necessary libraries and code to reproduce the following output:



```
c:\Users\mtktsun\source\repos\Debug\Task1.exe
Program Start
Press any key to continue . . .
```

### Task 2 (7 marks)

Implement a C++ class to represent a generic Unit entity in your game. Create both the header and implementation files for this class (Unit.h and Unit.cpp) inside the same project in Task 1. The unit must contain the following attributes (use appropriate data types, structures and access modifiers):

- a) fOwner – the identity of the owner (Human Player 1~n or Computer 1~n)
- b) fUnitCat – Category of the unit (Enum: LAND, AIR, WATER, BUILDING, TERRAIN)
- c) fID – unique identifier for this unit
- d) fPosition – the unit's position in the 2D game world
- e) fMaxHP – the unit's total health
- f) fCurrentHP – the unit's current health
- g) fMaxShield – the unit's maximum shield
- h) fCurrentShield – the unit's current shield
- i) fMaxEnergy – the unit's maximum Energy
- j) fCurrentEnergy – the unit's current Energy
- k) fGameState – indicate the unit's current state (Enum: PASSIVE, ACTIVE or DESTROYED)

Methods:

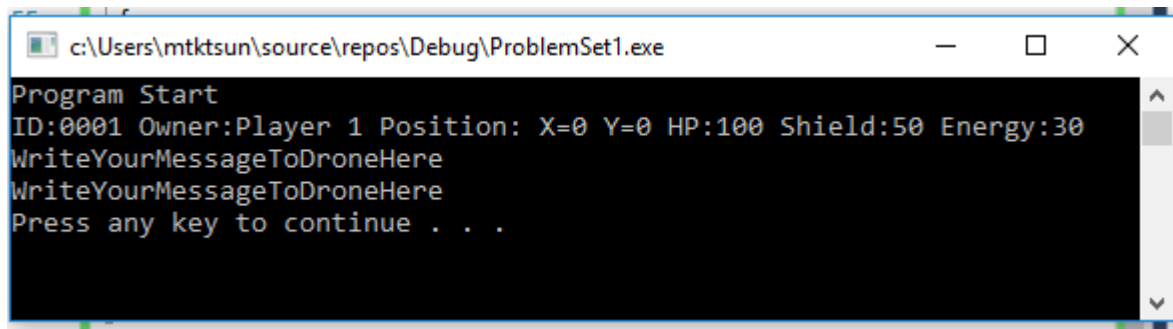
- a) Default Constructor – Creates an empty Unit. Initializes all members.
- b) Overloaded Constructor – Populate the attributes during Object creation.
- c) Accessors for each attribute – Getters and Setters (Get\_ and Set\_).
- d) Destructor – Used to release memory upon Deletion of this unit.

### Task 3 (9 Marks)

Continue to do the following in the ProblemSet1 project:

- a) Include the Unit class in Main.cpp.
- b) Add a string type attribute to the Unit class to store received messages: "fMessage".
- c) Add a friend overload of the Insertion Operator which saves the content of messages into "fMessage" when used.
- d) Add a friend overload of the Extraction Operator which outputs the content of "fMessage" when used.
- e) Add a public accessor function called "PingStatus" that outputs the contents of its attributes in a string (ignore the enumerated attributes if not possible).
- f) Create a Unit object called "Drone", and populate its attributes using its constructor or accessor functions.
- g) Use the Insertion Operator to save a message into "Drone".
- h) Use "PingStatus" to output "Drone's" status via the Console.
- g) Output the message from Drone to the Console using the Extraction Operator.

Your Console Output should look like this:



```
c:\Users\mtktsun\source\repos\Debug\ProblemSet1.exe
Program Start
ID:0001 Owner:Player 1 Position: X=0 Y=0 HP:100 Shield:50 Energy:30
WriteYourMessageToDroneHere
WriteYourMessageToDroneHere
Press any key to continue . . .
```

#### Task 4 (2 Marks)

Demonstrate how you can apply OOP Inheritance by extending the Unit class into 2 different derived classes. (Hint: you can derive one for a HeavyTank, CommandCenter, Trooper, etc).

#### Task 5(CHALLENGE: 5 Marks)

A spawned Unit cannot see or manipulate the attributes of other Units directly (thanks to encapsulation), but they can communicate to each other using messages.

E.g. In a DOTA 2 game session, Sniper shoots Pudge. Programmatically the Sniper Unit messages the Pudge Unit: "damage 50". Internally, the Pudge Unit will process this message and determines that it is instructed to deduct his Current Health by 50 points.

- a) Modify the overloaded Insertion Operator implementation to accept multiple parameter inputs.
- b) Replace "fMessage" or add additional attributes to capture Commands from the Insertion Operator.

Example Commands:

Command Message	Effect
Damage <Value>	Deduct CurrentHP by <Value>
Heal <Value>	Add CurrentHP by <Value>
Move <Value> <Value>	Increments Pos.X and Pos.Y appropriately

c) Modify your Main code to run in a loop until you choose to terminate the runtime. The loop should:

- i) Display the status of the Unit.
- ii) Prompt for Command Message.
- iii) Parse the Command and update the Unit's attributes accordingly.
- iv) Repeat the process unless an exit condition is met.