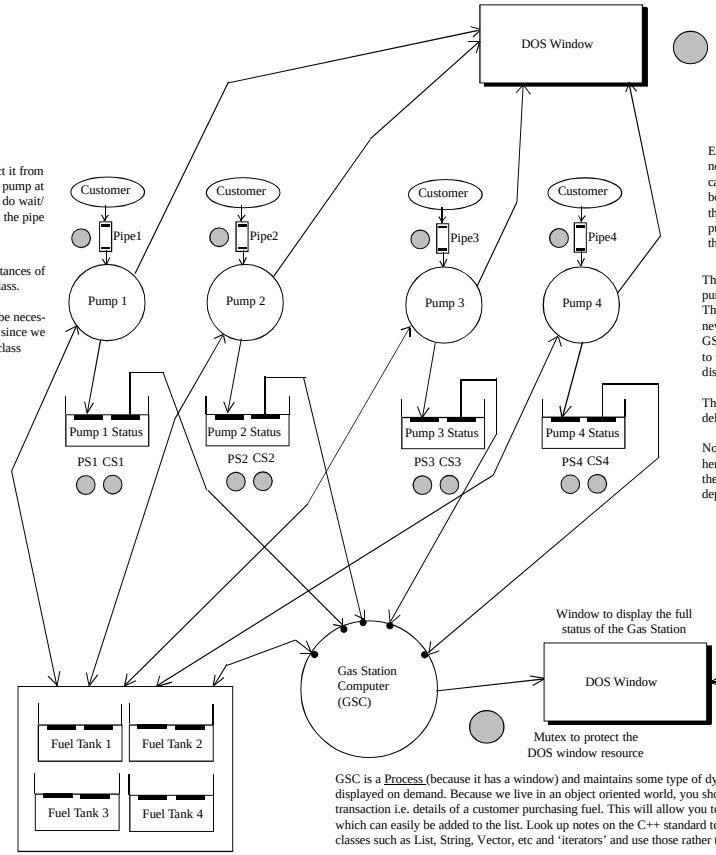
You should implement a **producer/consumer** arrangment between each thread in the Pump IO process and its respective Pump process (e.g. Pump 1). This will enable the Pump IO thread to know when the Pump process has updated its data pool. Hence use of semaphores ps1, cs1, ps2, cs2, ps3, cs3, ps4 and cs4

NOTE: Pumps 1-4 should be created as part of 1 single process, since only 1 window is needed.

1 Window to display the details of each customer (name, credit card etc) plus real time display of gas, bill, grade of fuel etc for ALL 4 PUMPs.



Mutex to protect the DOS window resource.
Mutual exclusion will be needed between the 4 Pump threads in order to write to the DOS window (hint use MOVE_CURSOR() in the rt library, printf() not cout, and call **fflush(stdout)** after each printf() call to force the OS to write the text to the window rather than buffer it up until a newline character is sent).

mutex for each pipe to protect it from multiple customers using pump at same time. Customers should do wait/ signal before writing to the pipe

Each of the 4 pipelines Pipe1-Pipe4 are TYPE-SAFE pipelines (see notes) which can hold a copy of a customer details, i.e. name, credit card, type of fuel required etc. The pipeline facilitate communication between a customer and a pump (both active objects with their own thread). The mutexes are here to ensure customer queueing at the pump should more than one customer try to use (and hence write to) the same pump at any one time.

Pump1-4 are active objects, i.e. instances of classes derived from ActiveClass.

When writing these it should only be necessary to write one class for a pump, since we can create 4 instances of that class

The four data pools exist to allow the pump to communicate details of a purchase to the Gas Station Computer (GSC)
The idea is that when a customer drives up to a pump, the details of the new transaction (e.g. name, credit card etc) are sent by the pump to the GSC for authorisation via the datapool. The GSC then communicates back to the pump (via a variable in the datapool) indicating that fuel can be dispensed.

The pump then keeps the GSC updated in real time with details of fuel delivery, grade of fuel etc and when the transaction has been completed.

Note that Pump and GSC implement a producer consumer arrangement, hence the need for the semaphore PS1, CS1 etc for each pump. Note that the Pump and the GSC can be a producer or a consumer at any time depending upon who is communicating with whom.

Window to display the full status of the Gas Station

2 character Keyboard Commands from Gas station attendent, such as refill fuel tank (e.g 'R' + F') and 'F' + '1' to start filling Pump 1 etc. Feel free to add other 2 charc commands/features as appropriate. Make sure you mouse click on this window before entering the commands otherwise they won't get sent to the GSC process.

Mutex to protect the DOS window resource

GSC is a Process (because it has a window) and maintains some type of dynamic data structure (e.g. a list) that can be accessed and displayed on demand. Because we live in an object oriented world, you should ideally introduce a C++ class to record a Gas station transaction i.e. details of a customer purchasing fuel. This will allow you to create 'transaction objects' that contain these details and which can easily be added to the list. Look up notes on the C++ standard template library (STL) on the course web site for built in classes such as List, String, Vector, etc and 'iterators' and use those rather than create your own.

GSC should have 5 child threads, 1 to handle the communication from each of the 4 pumps (via datapools 1-4 and semaphores) and another to check the status of fuel in each of the 4 Fuel tank. Black 'dots' inside the GSC process correspond to these threads. Each thread can write to the DOS window (a mutex will be needed to enforce mutual exclusion). Each thread can use, move_cursor(), printf() and fflush(stdout) etc to write to the window.

Fuel tanks to be created around the concept of a Monitor, with built in synchronisation (i.e. a mutex) and suitable interface functions to allow PUMPs to do things like decrement tank 1 by 1 litre, set new value for tank2, read tank3 value etc (add functions as you see fit)