TAU Advanced Topics in Programming - 2020B - Exercises

Amir Kirsh, Adam Segoli Schubert, Saeed Esmail

Requirements and Guidelines

The exercises in the course would require you to implement a stowage model for ships and a simulation to check stowage models.

If this is the first time you encountered the word “stowage” don’t be afraid, it’s more in use in the shipping domain. However, at the end of the course you might feel like a real shipmaster!

The exercise deals with organizing containers on a container ship.
To make things simpler, it will be based on some assumptions which are not exactly like the real world, but are close enough.

To get a glimpse into the domain of container ships, you can read and watch the following:
https://en.wikipedia.org/wiki/Container_ship
https://en.wikipedia.org/wiki/Stowage_plan_for_container_ships
https://www.youtube.com/watch?v=kj7ixi2lqF4
https://www.youtube.com/watch?v=DY9VE3i-KcM&t=1m55s

General instructions

Ship plan

A container ship has decks, or “floors”, starting with the lower floor, indexed zero.
Each floor has a “plan” which is like a map of the floor.
An upper floor may add new container spots that were not available in a lower floor, but it cannot remove spots that were available in a lower floor.
We assume all containers are the same size and sit exactly one top of the other.
The ship’s “plan” is the plan of all its floors.

Container

A container has the following properties:

- Weight in kgs (int)
- Destination port (5 english letters of seaport code)
- Unique identifier (as described by ISO 6346)

Ship Route

A ship has a finite route, containing any number of destination ports (5 english letters of seaport code), the route may have the same port more than once - in which case a container that is *on the ship* with destination port X ***must*** be unloaded on the first time the ship docks at port X.

Cranes operation

The dock cranes can pull any container if there is no other container on top of it. Cranes can lift and place containers at any given depth, even if a container is completely surrounded by other containers at any height.
The “cost” of pulling a container, at any location, is the same.

Stowage Process

When a ship reaches a port:

- it *provides* its “plan” as an input to the port’s crane management system.
- It *gets* as an input, the information about all containers that are about to be loaded to it.

The ship’s computer then needs to provide to the port’s crane management system a list of consecutive operations for loading and unloading so that all the containers that should be unloaded at this location would be unloaded and all the containers that should be loaded would also be. It is allowed to unload containers whose destination is not this port and then load them again (probably to another location). The goal of the ship’s computer is to optimize the number of load/unload operations over the entire route.
Note: in this exercise the ship gets the input of the containers to be loaded at each port. This allows only certain optimizations, keeping some of the input unknown (later loading inputs). In reality most of the loading information, if not all of it, is known in advance for the entire route.

Information provided for the ship on containers to be loaded:

list of *Container data* as described above

Information provided by the ship to the port's crane management system:

list of *load / unload* operations

Load / Unload operation data -- provided by the ship to the crane management:

"Load / Unload / Reject", container id, floor index, row index, column index

Note 1: the "Load / Unload" piece of data is redundant, as both sides can identify based on container id whether the operation is load / unload, however this information is still part of the protocol and serves as a safe control for data validity.

Note 2: "Reject" would be given for containers with a destination that isn't in the rest of the ship's route, or any other issue such as bad id. Another reason might be that the ship is full (you should prefer to reject later route destinations in this case) or weight restrictions seem to disallow this container at any location (see below). In the case of "Reject", the location indices would all be negative and may indicate the reject reason.

Weight balance

At any point of time, including when the ship is docking, the ship must keep weight balance. For this purpose the ship's computer has a balance calculator that takes into account weights and locations of all current containers and then for each set of future load / unload set of operations can provide the index of load / unload operation that would take the ship off balance.

Before passing the load / unload plan to the port's crane management system the ship must make sure that the plan fits weight balance limitations.

Parts of the exercise

Simulation

A program that loads data from files and then runs several stowage algorithms on the same data to compare the efficiency of the algorithms. It should also be possible to compare algorithms on several "travels" (different routes and cargo). Of course the simulation should also check that stowage algorithms are correct, i.e. they do not miss containers loading or unloading and follow all the rules in general. The simulation would also initiate the ship with its weight calculator, so it can control simulation of weight triggers and check that the stowage algorithms comply.

Stowage Algorithm

The stowage algorithm should be externally initiated with the following information:

- ship plan
- ship route
- weight balance calculator

We assume that the ship starts empty and gets its first cargo at the first port in the route.

Per each port stop, the algorithm shall comply to the following function calls:

- *getShipPlan()* - should return the ship plan (with no cargo information!)
- *getInstructionsForCargo(<containers to load in this port>)* - should return the set of load/unload instructions

Submission and Environment instructions

You will get submission and environment instructions separately. In short: the exercise should be implemented to run on Linux environment with standard C++ libraries. The exercise will be run and texted on Nova.

Exercise 1

In this exercise you are required to propose an API design and implement it fully, but with naive or stub implementations for some of the components.
Try to make your API design ready for changes since in the 2nd exercise we will publish an official API that you will be required to be synchronized with.

Why the hassle? Why didn't we publish a common API to begin with?

- We want *you* to practice the design of an API
- We want to pick the best API approach (or a mix) as the common API
- Then we want everybody to be synchronized with that common API
- We want you to practice *refactoring* - changing code while keeping functionality

Note that in this exercise you may have some open questions. If the questions are on the *story* - open a new topic in the course forum (questions on exercise 1). If the questions are on the design of the solution: in this exercise this is your ballpark, you should decide on a design that fits the story. You may still ask design questions in the forum, but such questions should include your thoughts and considerations or why you hesitate to take a certain approach.

Here are the parts of the exercise that you are required to implement in exercise 1:

1. Data structures, classes, input and output, input files and output files: you should decide on all of that
2. Exact names (of classes, methods that didn't get an exact name in the requirements, files, file extensions, etc.) -- is also your call
3. Simulation:
    a. Run a "cartesian loop" of "travel" *X* "algorithm"
    b. The weight calculator should have the proper API but can be naive and approve everything.
    c. Data to be collected by the simulation: you should decide
    d. Handling of error cases: you should decide what to do. Error cases (or any input) shall never lead to a program crash and should not be treated as OK (i.e. ignored). Example of error case: destination of container not in ship's route
4. Stowage Algorithm: you are required to implement a single algorithm that for each port first unloads all containers aimed for that port (if needed, first unload containers on top that prevent the crane from accessing lower containers - but remember to load them back!), then loads all waiting containers to arbitrary locations.

Good Luck!