**TAU Advanced Topics in Programming - 2020B - Exercises**

Amir Kirsh, Adam Segoli Schubert, Saeed Esmail

**Exercise 3 (link to** [**Ex1**](https://docs.google.com/document/d/1PloUy0vfoR0AWHmpWb0W85FZXM_BZqbaQhKXzHhwC1o)**, link to** [**Ex2**](https://docs.google.com/document/d/1qpXBOM24UiHUc5l2QiUlNGZOnfNm40aVd7bXbNVLIFs)**)**

# **Requirements and Guidelines**

In this exercise you should:

1. ~~Implement a WeightBalanceCalculator and adjust your Algorithms to react to reject responses from the WeightBalanceCalculator.~~
2. Improve your stowage Algorithms, for the traditional *Advanced Topics in Programming - Course Tournament*.
3. Allow your Simulator to run with multiple threads ~~and restrict the allowed time for algorithms’ run.~~
4. ~~Handle in relevant places maximum weight restriction for containers.~~

**Note**:

* The exercise is based on the requirements of Ex1 and Ex2. Any requirement from the previous exercises is still relevant, even if we didn’t check it in your previous submissions. Of course if there is a change in one of the requirements, the later exercise rules (i.e. Ex3 may override or change Ex2 or Ex1 requirements, the same as Ex2 may override or change Ex1 requirements).
* Testing is sometimes underestimated. Make sure to test your project properly before submission. Leave enough time for testing and bug fixes.



## 

## **WeightBalanceCalculator**

This requirement is **removed - keep your WeightBalanceCalculator as in Ex2!**

~~The calculator shall preserve the state of all loaded cargo of the Algorithm-Travel pair, either by managing it by itself or by using a “view” managed by the Simulator. Of course duplication of data if not necessary is not encouraged, so if the Simulator is already managing the required data and the data can be viewed by both the Simulator and the calculator this can be a totally valid solution.~~

~~The calculator shall have the following method:~~

~~BalanceStatus tryOperation(char loadUnload, int kg, int x, int y);~~

~~The method gets:~~

* ~~char: L or U for Load or Unload~~
* ~~kg weight~~
* ~~x and y position for the load / unload - the calculator knows the floor to load to (lowest possible) or to unload from (upper available container)~~

~~The method returns an enum value that can be one of:~~

* ~~APPROVED - operation approved (and registered as done by the balance calculator!)~~
* ~~X\_IMBALANCED - operation declined - need a better balance on the X coordinate~~
* ~~Y\_IMBALANCED - operation declined - need a better balance on the Y coordinate~~
* ~~X\_Y\_IMBALANCED - operation declined - need a better balance on both X and Y~~
* ~~EXCEEDS\_MAX\_WEIGHT - operation declined, container is too heavy~~

~~Note:~~

* ~~The balance calculator doesn’t have a “move” operation, as for balance purposes “move” is actually an “unload” followed by “load”. First the “unload” has to be approved, then the “load”.~~
* ~~We will refer to any IMBALANCED or EXCEEDS\_MAX\_WEIGHT result as a “reject” by the balance calculator.~~

~~In this exercise, the Simulator shall provide a concrete actual implementation of a real WeightBalanceCalculator and the Algorithm shall anticipate a certain interface, so there is a need to have an abstract base class for the communication between the two. The header of this abstract base class is defined below, inside WeightBalanceCalculator header file:~~

~~// WeightBalanceCalculator.h~~

~~#pragma once~~

~~#include <string>~~

~~class WeightBalanceCalculator {~~

~~public:~~

~~virtual ~WeightBalanceCalculator() {}~~

~~constexpr static int max\_weight\_kg = 25000; // container max weight = 25 ton~~

~~enum~~ *~~BalanceStatus~~* ~~{~~

~~APPROVED, X\_IMBALANCED, Y\_IMBALANCED, X\_Y\_IMBALANCED, EXCEEDS\_MAX\_WEIGHT~~

~~};~~

~~virtual~~ *~~BalanceStatus~~* ~~tryOperation(char loadUnload, int kg, int x, int y);~~

~~};~~

~~Note that the interface doesn’t have any init / readShipPlan method - as this is not part of the interface between the simulator and the algorithm. The actual calculator implementation may probably need an init operation, but not necessarily readShipPlan.~~

**~~Simulator’s Travel-Algorithm initiation flow:~~**

~~Per each Algorithm-Travel pair the Simulator would perform the following initiation flow:~~

1. ~~Read input files to see that the travel is valid~~
2. ~~Create WeightBalanceCalculator instance and initiate it~~
3. ~~Create the algorithm using its factory~~
4. ~~Initiate the Algorithm with:~~
   1. ~~ship plan~~
   2. ~~route~~
   3. ~~balance calculator~~

~~Note that any change in above order that is transparent to the outside world is allowed. (Order of Travel error messages, in case of errors, is not important and doesn’t dictate the order of initialization, as long as general travel errors come before Algorithm errors).~~

**~~WeightBalanceCalculator model and flow:~~**

1. ~~The actual implementation, as part of the Simulator project, shall model the containers on ship, either by using a view managed by the Simulator or by having its own model.  
   Of course duplication of data if not necessary is not encouraged.~~
2. ~~For each call to the method tryOperation, the WeightBalanceCalculator assumes that the required operation was done if APPROVED and not done if not (i.e if rejected).  
   This would of course influence all next WeightBalanceCalculator decisions. To model this information the WeightBalanceCalculator may propagate information to the Simulator or have its own model inside WeightBalanceCalculator. Of course duplication of data if not necessary is not encouraged.~~

**~~WeightBalanceCalculator logic~~**

~~You shall implement the following logic into your WeightBalanceCalculator:~~

~~Each of the axes, axis X and axis Y, is divided into an exact same number of rows / columns.~~

~~For example: if there are 5 rows on the Y axis and 4 columns on the X axis, then Y would be divided into: (0, 1) - 2 - (3, 4) and X would be divided into (0, 1) - (2, 3). In this example, row number 2 on the Y axis is exactly in the middle thus doesn’t affect any of the sides.~~

~~The~~ **~~difference~~** ~~of the~~ **~~total weight~~** ~~of~~ **~~each side~~**~~, across each of the axes separately, shall not exceed a certain threshold. In your implementation the threshold is:~~ **~~120 ton~~** ~~for the X axis and~~ **~~200 ton~~** ~~for the Y axis. Try to avoid “hard coding” these numbers in your code (defining constants in a proper place in your code is valid).~~

~~Note that for this exercise the definition of what is the X axis and what is the Y axis is NOT based on which of the two is smaller, we always assume that the Y axis is from Bow to Stern. Though, no need on your side to do anything special with this information as you just get X and Y info and use it.~~

~~Above is not exactly a real life logic, but it is the logic we ask you to implement (~~**~~so you cannot trade it for a “better” one~~**~~). However, our simulation can come with another logic and your Algorithm shall be ready for that. The only assumptions the Algorithm can make regarding the WeightBalanceCalculator are that (a) it is consistent; and (b) it follows reasonable physics laws (i.e. if taking off load from a given location takes the ship off balance, adding load above this same location, if not above valid floor, as well as taking off load from an opposite location -- both would be approved -- unless it offsets the other axis).~~

~~Note that the Algorithm is~~ **~~required~~** ~~to call the WeightBalanceCalculator for approval~~ **~~for any action~~**~~, even if it can reasonably assume that the action would be approved.~~

**~~Simulator’s validation of Algorithm’s compliance with WeightBalanceCalculator:~~**

~~The Simulator MUST check that the instructions output file generated by the Algorithm corresponds to the calls it made to the WeightBalanceCalculator, that there are no missing calls (i.e. the Algorithm didn’t skip calls to WeightBalanceCalculator) and that the value returned by the WeightBalanceCalculator was used correctly by the Algorithm (i.e. the instructions file matches the results returned by the WeightBalanceCalculator - approved actions were transformed into instructions and rejected actions were not).~~

~~In case the Algorithm didn’t play correctly, a proper error would be generated by the Simulator into error file and the result for this Algorithm-Travel pair would be -1.~~

**Update to the Score formula:**

The score formula would now count the following:

* Each **Load / Unload** operation would count as **5**
* Each **Move** operation would count as **3**
* **Reject** line in the crane instructions file - **doesn’t count**
* **~~Call to WeightBalanceCalculator~~** ~~that was~~ **~~approved~~** ~~-~~ **~~doesn’t count~~**
* **~~Call to WeightBalanceCalculator~~** ~~that was~~ **~~rejected~~** ~~would count as~~ **~~1~~**

Of course the score count shall be managed by the Simulator.

**~~New Algorithm Errors:~~**

~~2^20 - weight balance calculator not set (cannot run this travel)~~

~~2^21 - weight balance calculator bogus reject (following the reject, but we may be at infinite loop)~~

~~2^22 - container exceeds max weight allowed (ID rejected)~~

**~~Error 2^20~~** ~~is relevant in case the Algorithm didn’t get a WeightBalanceCalculator.~~

**~~Error 2^21~~** ~~is relevant if the WeightBalanceCalculator returns bogus results, i.e. rejects an unload from the same location where a load was already rejected just before or rejects load/unload where a load/unload from an opposite location was already rejected just before.~~

**~~NOTE:~~** ~~error 2^21 is optional your algorithm may decide to return it or not. In any case the Algorithm would follow the WeightBalanceCalculator result, even if it seems bogus.~~

**~~Error 2^22~~** ~~is relevant in case of a container that exceeds max weight. Simulator passes the file as is, but should issue an error and check that the algorithm rejects the container.~~

**~~Algorithm time limit:~~**

~~Each Algorithm-Travel pair would have a time limit that would be enforced by the Simulator. The simulation would have an additional~~ *~~command line parameter~~*~~:~~

**~~-travel\_time\_limit\_s <num>~~**

~~The value of this argument would set the time limit in seconds for each Algorithm-Travel pair. In case the algorithm exceeds this limit, the Simulator would generate a proper error and the score for this Algorithm-Travel would be -1.~~

~~This parameter, as others, can appear at any location on the command line. In case not provided - there would be no time limit per run.~~

**~~New Simulation Errors:~~**

1. ~~Algorithm not following WeightBalanceCalculator return value or not calling WeightBalanceCalculator per action.~~
2. ~~Algorithm-Travel run exceeds time limit.~~

~~In both cases the result for this Algorithm-Travel pair would be -1.~~

## **Multithreading**

The simulation would have an additional command line parameter:

**-num\_threads <num>**

This parameter, as others, can appear at any location on the command line.

In case not provided - the Simulator would run with a single thread (i.e. would not spawn any new threads, as if **-num\_threads ==** 1).

The **-num\_threads** parameter dictates the number of parallel threads for running the simulation. Note that the exact number of threads may be lower, in case there is no way to properly utilize the required number of threads. The exact threading model is your decision, try to make the most out of the provided threads.

## **Competition**

The competition is a bonus. In case you have a valid Algorithm that follows all the rules of Ex1 and Ex2, together with the new rules of Ex3 - you are good to go. However, you are still required to provide two somewhat different Algorithms.

The Algorithms that would reach the top positions in the competition would be eligible for a nice bonus (and TAU CS Pride and Glory Screenshot of “Advanced Topics 2020 Tournament Results”).

### Good Luck!