C++ Assignment

Candidate’s name:

Deliver your solution by:

The goal of this assignment is to design and implement the warehouse of a furniture factory. The stored products are sofas and chairs . The warehouse employs n workers and a single (1) packager. The job of the packager is simply to package products while the workers can move , lift and put down products.

The warehouse executes the following operations serving the factory’s back-office requests:

1. Inbound : Import a product in the warehouse.

The necessary tasks (steps) to inbound a product are: LIFT → MOVE → PUT_DOWN → MOVE A worker lifts the product, moves it to the appropriate position in the warehouse, puts it down and returns. 2. Deliver : Deliver a product to a customer.

The tasks (steps) for delivering a product are: MOVE → LIFT → MOVE → PACKAGE → MOVE → PUT_DOWN → MOVE A worker goes to the product, lifts it, moves it to the packager and the packager packs it. Then the worker moves it to the delivery point and puts the product down. Then she returns.

Back-office requests may arrive at the warehouse at any moment. If no worker is available at the time, the requests are queued and executed as soon as possible.

Each sofa product has a number of seats (integer), while each chair product has a weight (integer). The number of seats of a sofa is a random integer number in [2, 5). The weight of a chair is a random integer number in [1, 5). You can use getUniform (in utilities.h) to generate these numbers.

Each operation has an identifier, a type (INBOUND or DELIVER), the product type it refers to (SOFA or CHAIR), timestamps for the creation time, the start time and the finish time, as well as a list of the corresponding worker tasks .

Each task has an identifier, an operation identifier, a type (LIFT, MOVE, PACKAGE, PUT_DOWN), timestamps for the start and finish of the task, as well as a workload. The execution of each task is emulated by sleeping for workload milliseconds. Workload is a function of the product type of the corresponding operation. The formula that calculates the workload for each task type is:

● MOVE : 1

a. SOFA: 10 * number_of_seats * uniform[1,100) ms b. CHAIR: weight * uniform[1,100) ms

● LIFT:

a. SOFA: 3 * number_of_seats * uniform[1,10) ms b. CHAIR: weight * uniform[1,10) ms

● PACKAGE

a. SOFA: number_of_seats * uniform[20,30) ms b. CHAIR: uniform[20,30) ms

● PUT_DOWN

a. SOFA: number_of_seats * uniform[1,5) ms b. CHAIR: weight * uniform[1,5) ms

Back-office wants to keep a comprehensive record on the state of the warehouse. As a result, the warehouse is required to log when the tasks and the operations start and finish.

Task logs should follow the format below:

#- operation_id task_id start_time finish_time workload

And the operation report shall be in the following format:

## operation_id creation-time start-time finish-time

(see utilities.h for provided log functionality)

Implementation Details

Data input

To simulate the back-office requests, we provide the BackOffice class, that parses the input file and generates warehouse Requests (see Warehouse::onNewRequests function). The input txt file has the following format:

1For simplicity reasons, we assume that ALL moves are performed carrying the operation’s product. (Even if the last MOVE of each operation and the first MOVE of the Deliver operation are conceptually performed without any product.)

#number_of_workers IC DS ... DS 100

DS IC ... DS 10

DS DS ... DS 1000

Each line is a batch of back-office requests, followed by a wait period (in milliseconds) until the next batch. The requests are encoded as follows:

- IC: Inbound Chair
- IS: Inbound Sofa
- DC: Deliver Chair
- DS: Deliver Sofa

Provided functionality

In order to simplify the development, we provide the following functionality:

- CMake project
  - To execute the program after building run: ./warehouse input_file
- Logging (see logTask and logOperation functions in utilities.h file)
- Input file parsing (see BackOffice class)
- UUID generator (utilities.h)
- Random generator (getUniform function in utilities.h)
- Request struct (The backoffice requests)
- Warehouse class. This is the class where the required functionality should be added

● **main.cpp . The main function. Normally, no or minor edits are required here Feel free to modify, extend or ignore the provided functionality at will, as long as the requirements are met.**

Recommendations & Guidelines

- Use C++11. (Already configured in the given CMake project).
- The use of Boost libraries is recommended when necessary.
- Task execution is simulated by a usleep(workload * 1000) .
- Tasks are dependent to each other (must be executed serially).
- Operations are not dependent to each other (must be executed in parallel).
- The Packager is the only one that can PACKAGE products.
- The console output (log) must be readable .
- Your design should take into account that more products, tasks and operation could be added in the future.
- See sample_input.txt and sample_output.txt for the corresponding samples.